Dr. ing. **FLORIAN MORARU** • Dr. ing. **MIHAI ATODIROAEI**

PROGRAMAREA MICROCALCULATOARELOR ÎN SISTEMUL DE OPERARE CP/M

EDITURA ŞTIINŢIFICĂ ŞI ENCICLOPEDICĂ
BUCUREŞTI, 1989

FLORIAN MORARU • MIHAI ATODIROAEI

PROGRAMAREA MICROCALCULATOARELOR ÎN SISTEMUL DE OPERARE CP/M

Coperta: arh. CHRISTIAN NICOLESCU

ISBN 973-29-0024-5

CUPRINS

1. INTRODUCERE ÎN SISTEMUL DE OPERAT CP/M

Dintre toate tipurile de calculatoare utilizate și fabricate în țară ponderea cea mai mare ca număr o dețin microcalculatoarele cu microprocesoare 8080 sau Z80; această pondere va crește în următorii ani, datorită unui raport foarte bun între performanțe și cost față de alte sisteme de calcul.

Performanțele globale bune ale microcalculatoarelor pe 8 biți se datorează în primul rînd programelor folosite, care compensează performanțele mai slabe ale echipamentelor.

S-a ajuns de exemplu ca o compilare și • linkeditare de programe FORTRAN sau PASCAL să dureze aproximativ la fel pe un microcalculator ca și pe un minicalculator sau pe un sistem mai mare (de tip FELIX C-256/512), deși în cazul microcalculatoarelor memoria internă este de cel puțin patru ori mai mică, discurile au un timp mediu de acces de circa patru ori mai mare și o capacitate de zece ori mai mică, iar viteza procesorului este și ea de cîteva ori mai mică (în cazul microcalculatoarelor).

Un al doilea argument important în favoarea microcalculatoarelor îl constituie existența unui număr mare de limbaje de programare și a unor programe de aplicații moderne, care facilitează utilizarea acestor echipamente într-o diversitate de domenii.

Sistemul de programe CP/M este în același timp cel mai utilizat, dar și cel mai puțin cunoscut dintre sistemele folosite în țară. Numărul programelor utilizabile sub sistemul CP/M provenite din diverse surse depășește posibilitățile de cuprindere și de orientare ale utilizatorului obișnuit, totalizînd mii de pagini de documentație, ceea ce face aproape imposibil să se vorbească despre o documentație completă asupra sistemului.

Materialele documentare existente asupra sistemului CP/M sînt manuale de referință, care prezintă forma și efectul unor comenzi, fără însă să ofere recomandări privind utilizarea acestor comenzi sau exemple tipice de folosire a lor în situații reale apărute în practică.

De asemenea, sub CP/M se întîmplă deseori să existe mai multe programe care realizează aceeași funcție, iar utilizatorul trebuie să știe în ce situație sau în ce condiții trebuie folosit fiecare dintre aceste programe.

Consecința unei documentații incomplete și insuficient prelucrate este utilizarea ineficientă și necorespunzătoare a posibilităților sistemului CP/M și a microcalculatoarelor în general.

De aceea, vom insista în continuare asupra principalelor tehnici de programare și de operare folosite în cadrul sistemului CP/M, asupra limbajelor de programare considerate ca importante pentru aplicațiile uzuale ale microcalculatoarelor, folosind exemple simple dar extrase din programe reale.

1.1. Microcalculatoarele — structuri și utilizări

1.1.1. Echipamente cu microprocesoare

Într-un sens mai larg un microcalculator este orice echipament care folosește unul sau mai multe microprocesoare incluzînd de obicei și alte circuite integrate pe scară largă (circuite LSI).

Într-un sens mai restrîns un microcalculator este un sistem de calcul pentru dezvoltare de programe avînd unitatea centrală realizată cu microprocesor.

Se pot distinge următoarele categorii importante de sisteme cu microprocesoare:

— microcalculatoare pentru dezvoltare de programe și pentru aplicații diverse, dar de dimensiuni mai reduse decît cele realizate pe minicalculatoare sau pe sisteme mari;

— calculatoare personale pentru instruire și pentru petrecerea timpului liber;

— terminale programabile, cu microprocesor, cuplate la un alt sistem de calcul;

— echipamente de supraveghere și de conducere a proceselor și a unor sisteme diverse, care folosesc microprocesoare.

— scheme logice cu microprocesoare sau scheme de comandă cu logică programată.

Un microsistem pentru dezvoltare de programe are în configurație cel puțin o unitate de discuri (de obicei două unități de discuri flexibile) și o imprimantă, iar memoria modificabilă (de tip RAM) este de obicei între 48 și 64 kocteți.

Calculatoarele personale au o unitate centrală minimală și o tastatură, cu posibilități de cuplare la un televizor și la un casetofon, avînd de obicei facilități grafice și sonore care nu se întîlnesc în mod

normal la un microcalculator. Memoria ROM a calculatoarelor personale este de obicei destul de mare pentru a conține un interpretor BASIC și alte subprograme utile.

Terminalele programabile sînt tot microcalculatoare (eventual fără imprimantă), care dispun de una sau de cîteva interfețe de comunicație pentru linii seriale sau paralele prin care se pot lega cu alte sisteme de calcul

Echipamentele de control al proceselor cu microprocesor se caracterizează prin rezidența programelor în memoria fixă (de tip ROM) și dimensiunea redusă a memoriei RAM, prin absența dispozitivelor de imprimare și chiar a unităților de discuri și prin prezența unor interfețe speciale de proces pentru intrări-ieșiri numerice și analogice.

Schemele de comandă cu microprocesor sînt scheme logice de comandă a unor acționări electromecanice sau a altor dispozitive și fac parte integrantă din anumite echipamente. Logica de control este realizată printr-un program din memoria fixă ROM. Avînd un număr redus de circuite, sistemele cu logică programată sînt mai fiabile și mai flexibile decît schemele de comandă cu logică cablată, realizate cu componente mai puțin integrate.

Trebuie observat de la bun început că există o mare diferență între echipamentele cu disc și echipamentele fără disc din punct de vedere al utilizării și programării; microcalculatoarele cu disc reprezintă o clasă superioară față de cele cu casete magnetice sau fără memorie magnetică externă, indiferent de complexitatea unității centrale și a numărului de circuite LSI utilizate.

1.1.2. Familiile de circuite Intel 8080 și Zilog Z80

Microprocesorul 8080 este însoțit de o întreagă familie de circuite LSI, utilizate în principal pentru controlul dispozitivelor de intrare-ieșire sau pentru legătura cu sistemul în care este integrat microprocesorul.

Aceste circuite de interfață sau de control („controller" este un termen des folosit) se cuplează la porturile de I/E ale microprocesorului și se programează prin instrucțiuni de I/E.

Circuitele controlor de I/E din familia 8080 sînt destul de diferite ca funcții realizate, ca număr de porturi folosite, ca mod de programare, ca utilizare a sistemului de întreruperi etc.

În continuare vom trece în revistă pe scurt principalele circuite din familia 8080 utilizate și în microcalculatoarele de fabricație românească. Descrierea completă a acestor circuite se poate găsi în cataloage de firmă și în cîteva lucrări publicate care se ocupă de proiectarea cu microprocesoare.

Circuitul controlor de întreruperi 8259 asigură 8 niveluri de întreruperi de priorități diferite, fiecare nivel fiind asociat printr-o programare

inițială cu un controlor de intrare-ieșire. Unele echipamente de control a proceselor utilizează mai multe circuite 8259 pentru extinderea numărului nivelurilor de întrerupere.

Circuitul de ceas 8253 reunește trei contoare independente de cîte 16 biți, care pot fi programate să îndeplinească una dintre cîteva funcții posibile: generator de trenuri de impulsuri cu durată și frecvență programabile, divizor de frecvență, generator de impulsuri comandate prin program sau din exterior. Circuitul 8253 este folosit în principal ca ceas de timp-real al sistemului (pentru măsurarea timpului și pentru întîrzieri programabile) și ca divizor de frecvență programabil pentru controlorul de transmisie serială 8251.

Circuitul de acces direct la memorie 8257 (DMA) asigură patru canale separate de acces la memorie fără intermediul procesorului, canale utilizate de perifericele rapide, cu viteză mare de transfer: discuri magnetice, benzi magnetice, dispozitive de afișare cu ecran. Circuitul 8257 este cerut de circuitele de interfață pentru disc și pentru ecran (8271, 8272, 8275).

Circuitul pentru interfața serială sincronă și asincronă 8251 (USART) asigură controlul unei linii seriale de comunicație putînd fi programat pentru mai multe moduri de lucru, cu parametri diferiți și cu viteze de transmisie variabile. Uneori se folosesc mai multe circuite 8251 pentru a permite mai multe legături seriale, folosite fie pentru conectarea unui terminal video drept consolă, fie pentru o imprimantă, fie pentru legarea cu alte calculatoare sau terminale.

Circuitul pentru interfața paralelă de intrare-ieșire 8255 permite cuplarea microprocesorului fie cu diferite dispozitive de intrare-ieșire (tastatură, imprimantă, lector-perforator de bandă, lector de cartele, convertor N/A sau A/N), fie cu alte scheme electrice comandate de microprocesor. Circuitul 8255 poate fi programat astfel încît să funcționeze în paralel cu 8, 12, 16 sau 24 linii de date, precum și cu linii de control și de stare asociate unor linii de date. Utilizările curente ale lui 8255 sînt fie în echipamente dedicate, fie ca interfață paralelă de imprimantă.

Circuitul controlor de ecran 8275 permite afișarea de caractere alfanumerice și semigrafice pe un tub cinescop, cu anumite facilități de cursor programabil, de control a intensității și modului de afișare (direct/invers), de control a numărului de linii și de caractere pe linie etc. Circuitul 8275 este folosit în video-terminale (DAF, VDT ș.a.) și direct în cîteva terminale programabile (TDF, TPD, JUNIOR).

Interfața pentru unitățile de discuri flexibile de 8" sau de 5" se poate realiza fie cu circuitul 8271, fie cu circuitul 8272. Circuitul controlor de disc 8272 a înlocuit circuitul anterior 8271 în majoritatea echipamentelor, datorită avantajelor pe care le oferă: posibilitatea de a lucra și în dublă densitate (de fapt, cu mai multe densități diferite de înregistrare)

şi posibilitatea de a controla pînă la 4 unităţi de discuri (faţă de două, în cazul lui 8271).

Microprocesorul Zilog Z80 se poate utiliza ca înlocuitor direct pentru procesorul 8080, conectat cu circuite de interfaţă din familia 8080, sau se poate utiliza cu circuitele special proiectate pentru Z80.

Familia Zilog cuprinde 5 circuite de interfaţă, dintre care numai trei sînt utilizate în mod uzual: SIO, PIO, CTC.

Circuitele SIO, PIO, CTC se folosesc în modul propriu Z80 de tratare a întreruperilor, mod în care nu mai este necesar un circuit separat de control al sistemului de întreruperi. Sistemul de întreruperi este distribuit în unitatea centrală Z80 şi în circuitele de interfaţă, care se leagă într-un lanţ („daisy chain"); poziţia unui circuit în acest lanţ faţă de procesor determină automat prioritatea de întrerupere a circuitului respectiv. De obicei priorităţile sînt atribuite în ordine descrescătoare circuitelor de ceas CTC, de I/E seriale SIO şi de I/E paralele PIO.

Circuitul CTC (Counter/Timer Circuit) corespunde circuitului 8253, fiind utilizat ca ceas de timp-real (generînd întreruperi la intervale de timp prestabilite), ca divizor de frecvenţă programabil pentru circuitul SIO şi pentru numărarea unor evenimente externe. CTC asigură patru canale sub forma a patru numărătoare programabile independente.

Circuitul SIO (Serial Input/Output) corespunde circuitului 8251, fiind utilizat pentru controlul liniilor seriale de comunicaţie în mod asincron sau sincron. Circuitul SIO asigură două canale seriale independente şi multe facilităţi suplimentare faţă de 8251 pentru transmisia sincronă în diferite protocoale.

Circuitul PIO (Parallel Input/Output) corespunde circuitului 8255, fiind utilizat pentru cuplarea unor dispozitive de I/E diverse cu transfer paralel pe 8 biţi. Circuitul PIO asigură două interfeţe paralele cu cîte 8 linii de date şi 2 linii de control, care se pot programa separat în unul dintre patru moduri de lucru prevăzute.

1.1.3. Microcalculatoare fabricate şi utilizate în ţară

În prezent există o diversitate de microcalculatoare care folosesc microprocesoarele pe 8 biţi INTEL-8080 şi ZILOG-Z80, produse de mai multe întreprinderi din ţară, în serie mai mare sau mai mică. Multe dintre aceste echipamente au cunoscut mai multe variante, pe măsura dezvoltării şi perfecţionării lor constructive şi funcţionale.

S-au proiectat şi construit de asemenea cîteva microcalculatoare pe 16 biţi, cu microprocesoare INTEL-8088/8086, compatibile cu cel mai răspîndit calculator de acest tip IBM-PC (PC = Personal Computer). La Intreprinderea de Calculatoare Electronice (ICE) a intrat în producţia de serie un asemenea microcalculator sub numele de FELIX-PC

care folosește ca sistem de operare principal sistemul MS-DOS (firma MicroSoft), dar poate folosi și sistemul CP/M-86.

O altă mașină de același tip a fost realizată și la Intreprinderea de Echipamente Periferice București.

În textul care urmează ne vom referi numai la microcalculatoare pe 8 biți cu procesoare 8080 sau Z80 și de aceea vom folosi cuvîntul „microcalculator" cu sensul de microcalculator pe 8 biți.

Microcalculatoarele FELIX-M18, M18B, M118, M118B și CD-80 reprezintă prima familie de asemenea calculatoare și se produc încă din anul 1979 la Intreprinderea de Calculatoare Electronice din București.

Felix-M18 este un calculator complet compatibil cu sistemul de dezvoltare INTEL MDS (Micro Development System), care nu folosește circuite LSI din familia 8080, față de care are în plus unitate de bandă magnetică și un monitor extins pentru operații cu bandă magnetică. Banda magnetică constituie singurul suport compatibil cu sistemele de calcul Felix-C256/512.

Microcalculatoarele M118 și M118b folosesc controlorul de întreruperi 8259, circuitul de ceas 8253 și circuitul 8251, nu mai au bandă magnetică, au un monitor mai scurt și facilități grafice. Un monitor mai scurt înseamnă mai puțină memorie ROM și mai multă memorie RAM, deoarece pentru calculatoarele fabricate la ICE memoria totală adresabilă de 64 k este împărțită între memoria RAM (la adrese mici) și memoria ROM (la adrese mari).

Calculatorul CUB este o mașină 8080 cu multe circuite din familia acestui procesor, fără facilități grafice, dar cu un preț mult mai redus ca M118.

CUBZ este un calculator foarte compact, realizat cu circuite din familia Zilog Z80, avînd și facilități grafice la afișarea pe ecran (se pot afișa puncte individuale în afară de caractere alfanumerice).

Calculatoarele mai noi fabricate la ICE-M216, CUBZ, M118b— folosesc controlorul de disc integrat 8272 și pot lucra în dublă densitate.

M216 este o mașină duală, care folosește procesoarele 8080 și 8086, exploatabilă în sistemele de operare CP/M și MS-DOS.

Tot la ICE se fabrică și echipamentele pentru conducere de procese cu microprocesor 8080 din familia SPOT: SPOT-80, SPOT-83, fără discuri magnetice, dar cu interfețe de proces.

Calculatoarele personale HC-85, cu procesor Z80, compatibile cu Sinclair Spectrum, sînt produse la ICE și pot fi folosite împreună cu un televizor și cu un casetofon, iar mai recent și cu discuri flexibile.

Intreprinderea de Echipamente Periferice București (IEPER) a produs terminalele SIDM și TDF (cu 8080), iar în prezent fabrică terminalele programabile TPD și JUNIOR, cu o configurație de microcalculator. Anumite variante ale acestor terminale sînt grafice cu aceeași rezoluție ca M118 (512 × 256 puncte pe ecran).

Terminalul de pregătire a datelor TPD este un microcalculator care folosește majoritatea circuitelor din familia Intel 8080, inclusiv pentru ecran și disc.

JUNIOR este un calculator pe o placă realizat cu familia de circuite Zilog Z80 și cu controlor de disc 8272. Ambele mașini funcționează atît cu discuri de 8" cît și de 5" în simplă și dublă densitate și sînt prevăzute cu interfețe seriale și paralelă.

Tot la IEPER se fabrică și terminalul grafic DIAGRAM (DAF 2030) care folosește cîteva procesoare Z80.

Intreprinderea de Elemente pentru Automatizări București (IEA București) produce seria de echipamente de control și de automatizare ECAROM (880, 881), cu procesor 8080 sub forma unui sistem modular de plăci avind diferite funcții.

Intreprinderea IEA Cluj—Napoca fabrică terminalele programabile Telerom P286 (cu 8080) și P386 (cu Z80), cu discuri flexibile.

Intreprinderea Microelectronica din București fabrică microprocesorul MMN-80 (similar Z80), alte circuite din familia Z80 precum și sistemul pentru aplicații multiple MADS bazat pe Z80. Sistemul de plachete MADS include: modul unitate centrală, modul terminal grafic color, modul extensie memorie 64 k, modul programator de EPROM-uri, modul analizor logic, modul emulator Z80, modul cuplor de disc flexibil, modul de I/E numerice, modul achiziții de date analogice, modul interfață IEE-488 și placă universală pentru dezvoltare.

Intreprinderea de memorii Timișoara (FMECTC) a fabricat calculatorul personal AMIC și produce în prezent TIM-S, un calculator personal compatibil Spectrum, precum și mai multe mașini cu procesor 8080 și cu disc (MICRO-E ș.a.).

Pe lîngă echipamentele menționate s-au mai proiectat și s-au fabricat într-un număr mai mic de exemplare și alte microcalculatoare la diferite institute de cercetare și proiectare tehnologică din București, Cluj-Napoca, Pitești ș.a. precum și la instituții de învățămînt superior din București, Timișoara, Cluj-Napoca, Brașov, Craiova, Iași ș.a.

Marea majoritate a microcalculatoarelor existente au în configurație și o interfață pentru unități de discuri flexibile, ceea ce permite utilizarea unui sistem de operare cu disc.

Sistemul de operare SFDX-18 (echivalentul sistemului ISIS-II al firmei Intel) este utilizabil pe microcalculatoarele Felix M18, M18B, M118, M118B, M216 și Telerom P286; toate celelalte microcalculatoare cu disc folosesc sistemul de operare CP/M-80 (firma Digital Research).

1.2 Sistemul de operare CP/M-80

1.2.1. Sisteme de operare pentru microcalculatoare

Sistemul de operare este ansamblul programelor necesare utilizării unui calculator, deci necesare operării și programării echipamentului respectiv.

O primă împărțire a sistemelor de operare se poate face în funcție de suportul lor:

—sisteme de operare rezidente într-o memorie fixă și care nu necesită existența unei unități de disc;

—sisteme de operare cu disc.

Sistemele de programe rezidente în memorii ROM se folosesc fie pentru calculatoare personale fără discuri, fie pentru echipamente dedicate cu microprocesor, care execută mereu aceleași programe.

Sistemele de operare cu disc se folosesc la microcalculatoarele pentru dezvoltare de programe și pentru aplicații diverse, îndeplinind funcții similare sistemelor de operare de la sistemele de calcul mai mari, dar cu anumite simplificări, cerute de posibilitățile echipamentelor.

Un sistem de operare cu disc constă dintr-un nucleu rezident în memorie și o serie de programe memorate pe discuri, care se încarcă și se execută la cererea operatorului.

Nucleul sistemului se încarcă tot de pe disc la inițializarea sistemului și rămîne permanent în memorie, pentru că anumite funcții ale nucleului sînt necesare programelor care se execută (în principal funcții de intrare-ieșire).

Numărul și diversitatea programelor executabile pe un microcalculator cu disc sînt practic nelimitate.

Programele dintr-un sistem de operare cu disc pot fi împărțite în următoarele categorii:

— programe utilitare: editoare de texte, utilitare pentru operații cu fișiere și cu volume disc, depanatoare de programe, programe bibliotecar etc.;

— procesoare de limbaje: asambloare, compilatoare, editoare de legături etc.;

— programe de aplicații: procesoare de texte, sisteme de gestiune a fișierelor și bazelor de date, programe de calcul pe bază de tabele, programe pentru introducere de date, programe de comunicație etc.

Programele puse la punct sub un sistem de operare cu disc pot fi executate apoi tot sub sistemul respectiv sau pot fi încărcate în memorii ROM și executate pe echipamente specializate cu microprocesor, care nu dispun de discuri sau de alte periferice.

De obicei utilizatorul unui microcalculator este în același timp cel care programează și cel care operează microcalculatorul. Corespunzător acestor două ipostaze, un sistem de operare oferă o interfață sistem-

operator sub forma unui limbaj de comenzi consolă și o interfață sistem-programator sub forma unor funcții sistem direct apelabile din programe scrise în limbaj de asamblare și indirect folosite de programe scrise în limbaje evoluate.

Funcțiile sistem sînt în principal funcții de intrare-ieșire și pot fi utilizate numai în programele executate apoi sub același sistem de operare, deoarece în memoria calculatorului trebuie să existe, pe lîngă programul executat, și nucleul rezident al sistemului respectiv.

Pentru aplicațiile de supraveghere, de conducere și de comandă cu microprocesoare este utilă existența unui program monitor de timp real (numit și monitor de multitasking) care să asigure gestiunea mai multor taskuri paralele (concurente).

Într-o formă evoluată, programul monitor de timp real extins cu alte funcții și însoțit de alte programe utilitare specifice și de taskuri predefinite, devine un sistem de operare în timp real (de multitasking).

La microcalculatoare sistemele de operare în timp real sînt de obicei separate de sistemele pentru dezvoltare de programe, dar pot fi compatibile cu acestea la formatul fișierelor externe (pe disc sau pe alt suport). Taskurile executate sub sistemul de timp real sînt puse la punct (dezvoltate) sub sistemul de operare pentru dezvoltare de programe.

Pe microcalculatoarele românești se utilizează două sisteme de operare cu disc:

— sistemul SFDX-18;
— sistemul CP/M-80, în diferite variante.

Un sistem de operare pentru aplicații de timp real destul de răspîndit și de cunoscut este sistemul RMX-80 al firmei Intel, compatibil cu sistemul ISIS-II (SFDX-18).

Sistemul de operare MP/M reprezintă o extindere a sistemului CP/M cu funcții de multitasking, multiprogramare și multiacces, fiind în același timp un sistem de dezvoltare și un sistem de timp real.

Sistemul de operare CCP/M (Concurent CP/M) a fost dezvoltat tot de firma Digital Research, ulterior sistemului MP/M, dar nu este utilizat la noi în țară.

Folosirea eficientă a sistemelor MP/M și CCP/M este condiționată de existența unor configurații hardware care să includă memorii RAM de peste 64 de kocteți, precum și discuri rigide, de capacitate și performanțe superioare discurilor flexibile.

1.2.2. Prezentare generală a sistemului CP/M-80

Sistemul de operare CP/M-80 (Copyright Digital Research Inc) reprezintă încă de mai mulți ani un standard neoficial, instituit de practică, pentru microcalculatoarele cu 8080 sau cu Z80, în sensul că toate

calculatoarele cu disc construite pe baza acestor microprocesoare folosesc sistemul CP/M.

Succesul inițial al sistemului CP/M s-a datorat ușurinței cu care se poate implementa pe orice microcalculator cu 8080 sau cu Z80, ușurință explicabilă prin modul în care a fost conceput nucleul rezident al sistemului.

Mai precis, în CP/M a fost separată net o componentă minimală dependentă de echipament (numită BIOS = Basic Input Output System), singura care trebuie rescrisă pentru un nou microcalculator; celelalte componente ale nucleului sistem (BDOS și CCP), precum și aproape toate programele nerezidente (utilitare, asambloare, compilatoare, aplicații) pot fi utilizate fără modificări sau cu modificări minime pe orice echipament.

În consecință, putem vorbi despre o „mașină CP/M", care este o mașină cu 8080 sau Z80, pe care este implementat sistemul CP/M și pe care se pot folosi majoritatea programelor CP/M, deoarece particularitățile constructive ale mașinii sînt invizibile și deci neimportante pentru majoritatea utilizatorilor mașinii.

Performanțele superioare ale sistemului CP/M față de sistemul INTEL (ISIS-II) au constituit de asemenea un argument important în favoarea sa; reducerea timpului de lucru cu discul s-a obținut în CP/M prin acceptarea unei utilizări mai puțin eficiente a discului, realizată prin înlocuirea sectorului disc (de 128 de octeți) cu blocul (de 8 sectoare = 1 024 octeți) ca unitate de alocare a spațiului disc pentru fișiere.

Succesul inițial al sistemului CP/M-80 a fost amplificat apoi prin numărul mare de programe scrise de diverse firme și de utilizatori particulari, destinate acestui sistem și preluate pe diverse mașini CP/M.

În prezent există un număr impresionant de programe utilizabile sub sistemul CP/M, dintre care multe reprezintă modele de programe de calitate, fie din punct de vedere al facilităților oferite și al modului de utilizare, fie din punct de vedere al performanțelor acestor programe. Sînt disponibile astăzi sub CP/M practic toate limbajele de programare semnificative, cu implementări foarte bune, uneori de excepție (de exemplu produsul Turbo-Pascal).

Sistemul CP/M a fost conceput ca un sistem simplu și eficient pentru dezvoltare de programe, inițial numai în limbaj de asamblare.

Este un sistem de operare monoutilizator, conversațional și cu monoprogramare, cu o structură simplă a volumelor disc, cu un sistem de intrări-ieșiri și de fișiere deosebit de simplu.

Pentru a fi implementabil pe orice mașină, sistemul CP/M folosește un minim de resurse hardware: o consolă, un disc și eventual o imprimantă și alte două dispozitive de intrare și de ieșire (care inițial erau un lector și un perforator de bandă, astăzi mult mai rar utilizate).

Sistemul CP/M nu presupune existenţa unui sistem de întreruperi, a unui ceas, a unei console cu afişare (videoterminal), a unor legături seriale, deşi aceste facilităţi sînt comune în prezent tuturor microcalculatoarelor.

Sistemul CP/M poate fi adaptat să lucreze cu discuri diferite ca organizare şi ca performanţe; astfel pot exista diferenţe la numărul de capete, de piste, de sectoare pe pistă, de octeţi pe sector, la numărul de comenzi executate de către interfaţa discului etc.

La noi în ţară sistemul CP/M este implementat pe maşini cu discuri flexibile de 8" şi de 5", cu sectoare de 128, 256 şi 512 octeţi (cu simplă şi cu dublă densitate).

Maşinile CP/M cu discuri de 8" de simplă densitate (cu sectoare de 128 octeţi) sînt complet compatibile, în sensul că se pot citi sau scrie direct fişiere de la o maşină la alta. În dublă densitate pot exista anumite diferenţe între diferite maşini, deoarece nu există o convenţie unică relativă la mărimea sectoarelor disc şi nici la folosirea pistelor şi sectoarelor pentru nucleul sistem şi pentru directorul discului (tabela de fişiere a volumului).

În cursul evoluţiei sale sistemul CP/M-80 a cunoscut trei versiuni importante: 1.4, 2.2 şi 3.0. Pe microcalculatoarele româneşti se foloseşte versiunea 2.2. în diferite variante, dintre care unele includ anumite extinderi şi îmbunătăţiri ale performanţelor faţă de versiunea de referinţă a sistemului.

Sistemul CP/M a fost trecut şi pe maşini cu alte procesoare (CP/M-86 pe maşini cu Intel-8086, CP/M-68k pe maşini cu Motorola 68000), dar cu mai puţin succes decît pe microcalculatoarele pe 8 biţi. La microcalculatoarele pe 16 biţi s-a impus sistemul de operare MS-DOS, cunoscut şi sub numele de PC-DOS.

1.2.3. Programarea microcalculatoarelor sub sistemul CP/M

În cadrul sistemului CP/M se pot utiliza majoritatea limbajelor de programare uzuale, începînd cu limbaje de asamblare şi de macroasamblare, continuînd cu limbajele universale standard FORTRAN, COBOL, BASIC, PASCAL, C, PL/I, LISP şi terminînd cu limbaje mai speciale (de exemplu FORTH) sau cu limbaje specializate de nivel foarte înalt (de exemplu limbajul DBASE).

Memoria destul de limitată a microcalculatoarelor pe 8 biţi precum şi specificul multor aplicaţii ale acestor sisteme fac ca principalul limbaj de programare pentru maşinile CP/M să rămînă limbajul de asamblare.

În mod sigur, aplicaţiile de timp real şi programele frecvent utilizate sau care se încarcă în memorii ROM vor fi scrise în limbaj de asamblare, deoarece performanţele acestor programe constituie principalul criteriu de alegere a limbajului.

Programele executabile, rezultate din compilarea programelor scrise în alte limbaje, sînt de obicei mult mai mari decît cele scrise direct în instrucțiuni mașină (de circa 5—10 ori mai lungi, în funcție de limbaj, de compilator și de specificul programului).

Pentru programele cu multe calcule matematice (cu numere neîntregi) se va utiliza unul dintre limbajele FORTRAN, BASIC, C, PASCAL, toate incluzînd biblioteci aritmetice de calcul în virgulă mobilă și biblioteci de funcții matematice uzuale.

Aplicațiile de gestiune, care prelucrează fișiere de date, pot fi programate în DBASE, C sau COBOL, dar și în FORTRAN sau chiar BASIC. Prin facilitățile oferite, sistemul DBASE II asigură cel mai scurt timp de realizare a unei aplicații cu fișiere, dar timpul de execuție a programelor de aplicație este în general mare, datorită faptului că programul DBASE este un interpretor și nu un compilator.

Limbajul PL/I este un limbaj prea complex pentru mașinile pe 8 biți, iar timpul consumat în fazele de compilare-linkeditare este cel mai mare dintre toate limbajele compilate.

Limbajele PL/I, PASCAL, C, FORTH, LISP pot fi utilizate pe o mașină CP/M în scopul învățării acestor limbaje, familiarizării cu specificul lor, în vederea folosirii lor ulterioare pe alte calculatoare mai puternice.

Limbajele interactive, interpretate și nu compilate, cum sînt BASIC, LISP, FORTH, DBASE au avantajul unui timp de răspuns foarte scurt pentru utilizator între efectuarea unor modificări în program și constatarea efectului acestor modificări la execuție. Acest avantaj poate fi determinant în aplicații grafice sau în alte aplicații în care se experimentează în dialog cu calculatorul.

Același avantaj îl oferă compilatorul TURBO-PASCAL, care funcționează numai în memoria internă, fără să utilizeze discul, și care face o compilare aproape instantanee a programelor (la cerere, programele compilate pot fi scoase pe disc).

Dezvoltarea și punerea la punct a programelor în regim de interpretare sau de compilare în memorie, cu editor incorporat, au nu numai o productivitate mult mai mare față de modul de lucru clasic (editare-compilare-linkeditare-încărcare și execuție, cu fișiere intermediare între aceste faze), dar și o atractivitate sporită pentru utilizatori.

Trebuie menționat de asemenea că implementările limbajelor FORTRAN, PASCAL, COBOL, BASIC și C se conformează standardelor existente și de multe ori extind cu facilități utile versiunile standard (în acest sens o mențiune deosebită pentru PASCAL MT+ și pentru TURBO-PASCAL, care includ numeroase facilități).

Pentru aplicații mai mari, la care performanțele nu sînt critice și care trebuie realizate într-un timp cît mai scurt, alegerea se face între limbajele C, PASCAL sau FORTRAN în funcție de caracteristicile

aplicației și de avantajele specifice ale fiecărui limbaj, așa cum sînt ele implementate sub CP/M.

De observat că programele scrise în C (Aztec) și în PASCAL MT+ pot fi oricît de mari și pot fi segmentate relativ ușor, deoarece editoarele de legături folosite cu aceste compilatoare prevăd facilități de încărcare automată a segmentelor nerezidente, apelate din cadrul unor programe segmentate în mai multe fișiere disc.

1.2.4. Programe de aplicații în sistemul CP/M

Aplicațiile de gestiune cu volum mic de date, pentru care un microcalculator cu discuri flexibile este suficient, pot fi realizate într-un timp scurt folosind sistemul de gestiune a bazelor de date *DBASE II* (firma Ashton-Tate și firma RSP Inc).

Sistemul DBASE poate fi utilizat pentru aplicații de introducere de date, memorare-regăsire de date, raportare și prelucrare date.

Pentru introducere (colectare) de date pe suport magnetic de la consola unui microcalculator, prin completarea unui formular afișat pe ecran și conceput de utilizator, se poate folosi programul *DataStar* (firma MicroPro), împreună cu programul pentru definire de formulare FormGen și cu programul Batch de introducere de date dintr-un fișier.

Fișierele create cu DataStar sau cu alte programe pot fi sortate cu programul *SuperSort* (firma MicroPro), care include multe facilități utile asociate procesului de sortare-interclasare și căutare după chei multiple.

Multe aplicații contabile, de calcul și de optimizare pe baza unor foi de calcul tabelare pot fi rezolvate cu ajutorul programului *Multiplan* (firma MicroSoft), program din categoria „foaie de calcul electronică " („Spreadsheet").

Crearea, menținerea și imprimarea textelor și documentelor de orice fel se poate face deosebit de comod cu ajutorul programului *WordStar* (firma MicroPro), un editor-procesor de texte unanim apreciat pentru funcțiile pe care le oferă și pentru interfața deosebit de prietenoasă cu utilizatorii.

Toate programele menționate funcționează cu consola microcalculatorului în mod ecran, folosind anumite secvențe de control specifice fiecărei console (ștergere ecran, poziționare cursor etc.) și de aceea este necesară în general o „instalare", adică o configurare a programelor respective la caracteristicile videoterminalului utilizat. Această instalare a programelor se face fie printr-un program special de instalare, care însoțește programul de aplicație, fie prin modificarea programului cu ajutorul unui depanator (DDT sau SID), la anumite adrese indicate în documentația programului respectiv.

Programele de comunicație între calculatoare integrează două funcții importante: funcția de emulator de terminal și funcția de transfer de

fişiere; prima permite utilizarea unui microcalculator ca terminal simplu la un alt sistem de calcul mai puternic, iar a doua permite schimbul de fişiere (de date sau de programe) între calculatoare diferite sau de acelaşi tip, dar care nu sînt compatibile la discurile utilizate.

Programele de comunicaţie trebuie adaptate pentru fiecare microcalculator în parte, deoarece nu pot fi scrise numai cu funcţii de intrare-ieşire ale sistemului CP/M.

Dintre programele de comunicaţie existente şi folosite în ţară menţionăm produsele MOVEIT, KERMIT/TRAVEL şi CROSSTALK, dar numărul programelor din această categorie este mult mai mare.

Pentru aplicaţiile de timp real există mai multe programe de tip *monitor de multitasking* sub CP/M, dintre care menţionăm programul MTK-80 (Multi-Tasking Kernel) publicat de revista BYTE.

Verificarea gramaticală şi semnalarea erorilor din texte scrise în limba engleză pe baza unui vocabular extensibil se poate face cu ajutorul programelor *SpellStar* (firma MicroPro) sau *Spell*.

Aplicaţiile grafice ale microcalculatoarelor nu beneficiază de programe universale din cauza diferenţelor mari între facilităţile grafice oferite şi a modului cum ele se programează la diferite maşini, dar există mai multe biblioteci de subrutine grafice utilizabile din Fortran şi care pot fi relativ uşor adaptate pe fiecare maşină.

Acceptînd ca program de aplicaţie orice program direct utilizabil de către anumiţi utilizatori, se pot considera ca programe de aplicaţie şi programele de jocuri, în număr foarte mare. Din motivele menţionate jocurile grafice nu sînt însă uşor portabile.

1.3. Ghid de utilizare pentru sistemul CP/M

1.3.1. Interfaţa sistemului cu operatorul

Prima operaţie necesară pentru a utiliza un microcalculator este încărcarea de pe disc în memorie a nucleului CP/M, operaţie declanşată de obicei prin acţionarea unui comutator sau a unei taste sau prin simpla introducere a discului sistem în unitatea de discuri notată cu 0.

După încărcare, la consolă apare un mesaj care precizează versiunea de sistem cu care se lucrează şi memoria RAM pentru care este configurat sistemul; pe linia următoare se afişează de obicei următoarele două caractere: A>

Caracterul „prompter" '>' invită la introducerea unei comenzi de către operator, arătînd că sistemul este gata şi aşteaptă o comandă.

Litera 'A' reprezintă numele CP/M pentru unitatea de disc cu numărul 0 şi aminteşte utilizatorului că discul de lucru implicit este în acest moment discul A (unitatea 0).

În mod similar, unitatea de disc numărul 1 are în CP/M numele B, unitatea 2 are numele C ş.a.m.d.

Discul implicit curent se poate modifica oricînd prin introducerea numelui unităţii care urmează să devină disc curent, urmat de caracterul ':'. Exemplu:

A> B:<CR>
B>

De observat că orice comandă introdusă de la consolă trebuie terminată prin caracterul CR (Return), caracter neafişabil reprezentat uneori în textele care descriu comenzi prin <CR>.

În acest manual nu vom reprezenta în general caracterul terminator al liniei de comandă, presupus implicit în toate exemplele. De asemenea, nu se vor mai scrie caracterele afişate de calculator, ci doar cele introduse de utilizator (şi afişate în ecou de sistem la consolă).

Noţiunea de disc implicit este importantă, deoarece permite simplificarea referirilor la fişiere: pentru fişierele aflate pe discul implicit nu este necesară precizarea acestuia în specificatorul de fişier, dar pentru fişierele aflate pe alte unităţi este necesară specificarea unităţii de disc pe care se află aceste fişiere.

În practică se constată că majoritatea sau toate fişierele pe care un utilizator le foloseşte intens se află pe un acelaşi disc, care va fi desemnat ca disc implicit (cel puţin pentru o anumită perioadă de lucru).

Este recomandabil ca înainte de a folosi o dischetă să se facă o copie de rezervă, pentru ca în cazul distrugerii accidentale a unor fişiere sau a pistelor sistem să se poată reface discul stricat. Copierea pistelor sistem se face cu programul SYSGEN, iar copierea fişierelor se face cu programul PIP (dacă sînt două unităţi de disc), DIP (utilizabil şi pe o singură unitate), sau altele; de asemenea, există programe de copiere fizică a unui disc în întregime (pistă cu pistă).

Probabil că prima comandă folosită de mulţi utilizatori după încărcarea sistemului este o comandă de afişare a conţinutului discului (discurilor). Această comandă poate fi DIR sau STAT sau D sau numele altui program utilitar destinat acestui scop.

Comanda DIR, împreună cu alte cîteva comenzi uzuale, există pe orice disc sistem, chiar dacă acesta nu conţine nici un fişier, deoarece sînt comenzi rezidente în nucleul sistemului.

Orice altă comandă în afara celor 6 comenzi rezidente (DIR, ERA, TYPE, REN, SAVE, USER) este de fapt un nume de fişier (cu extensia COM), fişier care conţine un program direct executabil. De aceea, trebuie să aflăm mai întîi ce fişiere de tip COM avem pe discurile introduse pentru a şti ce comenzi (programe) putem utiliza.

Comanda DIR afişează la consolă numele şi extensia tuturor fişierelor aflate pe un disc, în ordinea înregistrării lor pe disc. Mai

precis, comanda DIR poate avea diferite forme, prin care se pot selecta discul suport şi un anumit grup de fişiere care se vor lista. Exemple:

DIR listare nume fişiere de pe discul implicit
DIR B: listare nume fişiere de pe discul 1
DIR *. COM listare nume fişiere de tip COM
DIR B: TEST.* listare nume şi extensie fişiere de pe discul 1 care au numele TEST

Caracterul asterisc *, folosit într-un specificator de fişier, are sensul de oricare (orice nume sau orice tip) şi se foloseşte pentru a desemna un grup de fişiere care au ceva în comun.

Ca un detaliu CP/M menţionăm că prin comanda DIR se afişează numai fişierele cu atributul DIR care au codul utilizatorului egal cu codul curent (nu se listează numele fişierelor care au atributul SYS şi care au alt cod utilizator). Trebuie spus că majoritatea fişierelor au codul 0 (în care se lucrează în mod curent) şi că au atributul de vizibilitate DIR.

Pentru a obţine şi spaţiul ocupat de fiecare fişier precum şi spaţiul rămas liber pe un disc sau pentru a vedea şi fişierele cu alte coduri sau invizibile la DIR, trebuie folosită una dintre celelalte comenzi (nerezidente): STAT, D, XDIR, POWER etc.

Comanda TYPE listează la consolă conţinutul unui fişier şi se poate folosi pentru fişierele de tip text (care conţin numai caractere afişabile). Exemple:

TYPE CPM.DOC afişare conţinut fişier CPM.DOC de pe discul implicit
TYPE B: TEST.ASM afişare program din fişierul TEST.ASM aflat pe unitatea 1

Afişarea unui text mai lung (care defilează pe ecran) poate fi oprită temporar cu caracterul CTRL/S, poate fi reluată apoi cu CTRL/S sau CTRL/Q, şi poate fi terminată cu orice alt caracter.

Comanda ERA (Erase) are rolul de a şterge unul sau mai multe fişiere de pe discul specificat. Exemple:

ERA*.BAK şterge toate fişierele de tip BAK de pe discul implicit
ERA B:TEST.* şterge toate fişierele cu numele TEST şi de orice tip de pe unitatea 1
ERA *.* şterge toate fişierele de pe discul curent

Cu excepţia ultimului exemplu, nu se cere confirmarea ştergerii şi nici nu se afişează numele fişierelor şterse. În CP/M este posibilă recuperarea unor fişiere şterse din greşeală, dacă nu s-au creat între timp alte fişiere pe acelaşi disc.

Comanda REN (Rename) permite schimbarea numelui unui fişier. Exemple:

REN CPM.TXT = CPM.DOC schimbă numele fişierului CPM.DOC în numele CPM.TXT

REN B: TEST2.ASM = B: TEST.BAK schimbare de nume pe discul B:

Comanda SAVE permite salvarea pe disc, într-un fişier, a unui program din memorie, cu indicarea numărului de octeţi salvaţi (exprimat în număr de pagini de 256 octeţi). Exemplu:

SAVE 2 TEST.COM scrie 512 octeţi de la adresa 100H

Comanda USER permite stabilirea sau modificarea codului utilizator în care se lucrează, cod care se ataşează fişierelor create. Codul este un număr între 0 şi 15. Exemplu:

USER 5 stabileşte codul curent la valoarea 5.

Comenzile rezidente se deosebesc de comenzile nerezidente prin aceea că se pot executa de sub orice cod, se execută mai repede şi nu necesită prezenţa anumitor fişiere pe disc; ele nu pot fi precedate de un nume de disc.

Se observă că toate comenzile rezidente de afişare (ca şi multe comenzi nerezidente) folosesc în mod implicit consola ca dispozitiv de afişare. Pentru a obţine o listare pe hîrtie, la imprimantă, a informaţiilor afişate pe ecranul consolei se poate folosi caracterul CTRL/P cu rol de comutator pentru activarea sau dezactivarea listării la imprimantă a tot ceea ce se afişează pe ecran. Caracterul CTRL/P poate fi introdus oricînd, înainte, după sau într-o comandă, avînd efect imediat.

Caracterele de control, neafişabile, se introduc prin apăsarea simultană pe clapa CONTROL şi pe o altă clapă; această acţionare simultană de două clape se notează în texte cu CTRL/P sau cu ↑ P, de exemplu. Efectul apăsării clapei CTRL simultan cu un alt caracter este acela de anulare a bitului 5 din codul caracterului respectiv, echivalent cu a scădea valoarea 40 hexa din codul său; de exemplu, litera P are codul ASCII 50H iar codul caracterului CTRL/P este 10 H.

Notaţia hexazecimală (în baza 16) se foloseşte ca prescurtare pentru numerele binare (în baza 2); vom folosi convenţia limbajelor de asamblare de a urma numerele hexa de litera H. Exemple:

0FH (15), 10H (16), 40H (64), 80H (128), 0FFH (255), 100H (256)

Comenzile nerezidente sînt nume de fişiere de tip COM, la care nu se mai specifică tipul fişierului, dar la care trebuie specificat numele discului suport, dacă diferă de discul implicit. Exemple:

DU încarcă şi execută programul din fişierul DU.COM
B:STAT execută programul STAT de pe discul 1

De multe ori comenzile de apelare a unor programe, ca şi comenzile rezidente, mai conţin şi unul sau mai mulţi parametri necesari execuţiei comenzii (nume de fişiere, opţiuni, numere ş.a.). Exemple:

B:DDT TEST.COM
PIP LST: = COM.DOC
COMPARE WS.COM B:WS3.COM
CII—M TEST.C.

Multe dintre programele CP/M (dar nu toate) au două forme de apelare: o formă fără parametri şi o formă cu parametri.

Forma fără parametri încarcă doar programul, urmînd ca datele necesare să fie introduse ulterior; această formă se foloseşte atunci cînd se solicită mai multe operaţii succesive programului apelat. Exemplu:

```
A>PIP
*B: = PIP.COM
* TEST2.ASM = B: TEST.BAK
* ↑ C
```

Forma cu parametri transmite datele necesare în linia de comandă; această formă se foloseşte pentru o singură operaţie solicitată programului apelat. Exemplu:

```
A>PIP B: = PIP.COM
```

Pentru a veni în ajutorul operatorului, programele care realizează mai multe funcţii pot afişa implicit sau la cerere o listă a acestor comenzi, cu sintaxa şi rolul fiecărei comenzi (de exemplu, programele POWER, DBASE, MULTIPLAN, WORDSTAR, DATASTAR ş.a.) sau unele indicaţii de utilizare (DIP, Q, UDISK ş.a.)

Din cele spuse pînă acum rezultă cîteva caracteristici ale sistemului CP/M, aşa cum este el văzut de la consola operatorului:

— în CP/M nu există un număr limitat de comenzi, pentru că orice nou program introduce în sistem o nouă comandă prin care se apelează programul respectiv;

— fiind nume de fişiere, comenzile nerezidente pot fi precedate de un nume de disc (A:, B:, C: etc).;

— în CP/M nu există o sintaxă unică pentru toate comenzile, practicîndu-se o diversitate de forme pentru transmiterea datelor necesare execuţării comenzii.

Această libertate în forma comenzilor provine din faptul că procesorul de comenzi consolă din CP/M (numit CCP) nu pretinde o anumită formă de introducere a parametrilor comenzii. CCP nu analizează decît parţial linia de comandă, pe care o transmite într-o zonă-sistem programului apelat; acesta poate analiza şi interpreta coada comenzii după propriile sale convenţii („coada comenzii" este şirul de caractere care rămîne din linia de comandă, după extragerea numelui comenzii).

Datorită capacităţii limitate a discurilor flexibile şi necesităţii de a citi sau scrie discuri „străine" (de la alte maşini), deseori se pune problema înlocuirii discului dintr-o unitate cu un alt disc (de aceeaşi densitate sau cu altă densitate recunoscută de sistem). Schimbarea discului se poate face în orice unitate, dar la unele variante de CP/M în unitatea 0 nu poate fi introdus decît un alt disc sistem pentru maşina respectivă.

În CP/M manevra de înlocuire a unui disc într-o unitate trebuie în general urmată imediat de tastarea caracterului CTRL/C, pentru ca sistemul să recitească structura noilor discuri. Mai exact, această operație este strict necesară dacă urmează o scriere (sau ștergere) de pe noul disc introdus, deoarece citirea se poate face oricum. Totuși, ca regulă generală este bine să se reinițializeze sistemul prin CTRL/C la fiecare schimbare de disc.

1.3.2. Dezvoltarea de programe în CP/M

După exersarea comenzilor rezidente și a unor programe utilitare, etapa imediat următoare este cea de introducere și de testare a unui program simplu într-unul dintre limbajele de programare cunoscute.

Diferențele în utilizarea unor limbaje diferite pot fi destul de mari, după cum se folosește un interpretor sau un compilatcr și în funcție de programele de sistem folosite.

Utilizatorul trebuie să cunoască un editor de texte precum și particularitățile de operare și de limbaj ale compilatorului sau asamblorului folosit, precum și ale unor programe asociate.

Vom prezenta aici operațiile necesare obținerii unui program executabil, simplu, dintr-un program-sursă scris în limbaj de asamblare, în cazul utilizării primului asamblor CP/M, numit ASM. Vom presupune că pe discul din unitatea A se află toate programele necesare: WM.COM, ASM.COM, LOAD.COM.

Prima operație este cea de creare a unui fișier-sursă care să conțină programul în limbaj de asamblare, realizată cu ajutorul unui editor de texte. În ipoteza că se folosește editorul în mod ecran WordMaster, atunci comanda de lansare a editorului este

```
WM TEST . ASM
```

sau

```
WM B: TEST . ASM
```

dacă fișierele de lucru se creează pe unitatea B.

După încărcarea editorului de texte în memorie, începe introducerea programului de la tastatură, folosind caracterele TAB pentru alinierea codurilor de instrucțiuni, CR (Return) pentru terminarea fiecărei linii și DEL (RUB) pentru ștergerea ultimului caracter introdus (la stînga cursorului).

Fie următorul program simplu, care afișează la consolă un caracter ('#'):

```
ORG 100H; adresa de încărcare și execuție
MVI E, '#'; caracter de afișat
MVI C, 2; cod funcție de afișare
CALL 5; apel sistem
```

```
JMP 0; terminare program şi salt în sistem
END
```

După introducerea ultimei linii (şi ea terminată cu ⟨CR⟩), se apasă pe tasta ESC (Escape) pentru a trece în mod comandă şi se introduce litera E pentru comanda 'Exit' de terminare a editării (litera E urmată şi ea de ⟨CR⟩). În acest moment programul introdus în memorie se va scrie pe disc.

Fişierul creat poate fi verificat prin afişare cu comanda

TYPE TEST.ASM sau TYPE B:TEST.ASM

Urmează faza de asamblare, lansată prin comanda

ASM TEST sau ASM B:TEST

La terminarea asamblării, marcată prin afişarea unui mesaj la consolă, s-au creat pe acelaşi disc încă 3 fişiere:
TEST.HEX, TEST.PRN şi TEST.SYM.

Putem verifica existenţa acestor fişiere prin comanda

DIR TEST.* sau DIR B:TEST.*

Fişierul TEST.PRN conţine o listă de asamblare, cu forma sursă, forma internă şi adresele atribuite instrucţiunilor; putem afişa această listă folosind comanda TYPE şi o putem tipări la imprimantă introducînd caracterul CTRL/P înaintea lui TYPE.

Ultima fază este cea de transformare a programului obiect din format hexazecimal (TEST.HEX) în format direct executabil, folosind programul LOAD:

LOAD TEST sau LOAD B:TEST

care creează un nou fişier TEST.COM.

Execuţia programului astfel obţinut se poate face prin comanda

TEST sau B:TEST

şi are ca efect afişarea pe ecran a caracterului '#'.

În cazul folosirii unui alt asamblor sau a unui alt limbaj aceste faze pot fi diferite. Detaliile de operare şi particularităţile de implementare a limbajelor disponibile în CP/M sînt prezentate în capitolele următoare.

2. ORGANIZAREA INTERNĂ A SISTEMULUI CP/M

Pentru majoritatea programatorilor și utilizatorilor unui sistem de operare este suficientă o cunoaștere din exterior a sistemului, a funcțiilor oferite de sistem și a convențiilor utilizate.

În cazul sistemelor de operare mai complexe nu este necesară și nici posibilă cunoașterea organizării interne și funcționării componentelor sale, a structurilor de date interne și pe disc.

Sistemul CP/M este total deosebit în acest sens, deoarece nu caută să ascundă utilizatorilor detaliile privind modul său de lucru și chiar necesită uneori cunoașterea unor asemenea detalii (cum ar fi, de exemplu, structura descriptorului de fișier FCB, organizarea discurilor și a fișierelor ș.a.).

Pe de altă parte, pentru a utiliza mai bine un sistem de operare, este utilă o informare chiar sumară asupra modului în care lucrează sistemul respectiv. Utilizatorul de la consolă sau programatorul trebuie să știe nu numai cum să procedeze, dar și de ce trebuie procedat astfel, care este explicația unor erori sau a altor situații neprevăzute.

În cazul sistemului CP/M această cunoaștere este nu numai necesară, dar și posibilă, chiar în detaliu, deoarece sistemul CP/M este probabil cel mai simplu sistem de operare dintre cele folosite în mod curent.

Eliminînd tot ceea ce nu era strict necesar, autorul sistemului CP/M a reținut numai funcțiile esențiale dintr-un sistem de operare, reușind să obțină un sistem simplu și performant. Avînd un nucleu rezident foarte compact, care ocupă în memoria internă și pe disc un spațiu minim, rămîne spațiu suficient atît în memorie cît și pe discul sistem pentru programele de aplicații, iar un program care dispune de memorie poate avea și performanțe mai bune (fiindcă se reduce sau se elimină segmentarea programelor mari și se pot folosi zone tampon mai mari pentru accesul la fișierele de date).

Comparat, de exemplu, cu sistemul RT-11 de pe minicalculatoare, sistemul CP/M are performanțe net superioare din toate punctele de

vedere în cazul utilizării unor unități de discuri similare, fără să se limiteze domeniul de utilizare sau facilitățile oferite utilizatorilor și programatorilor.

În acest capitol se prezintă o serie de informații necesare programării în limbaj mașină și utilizării sistemului (organizarea memoriei interne, sistemul de intrări-ieșiri, convenții ale sistemului de fișiere și ale procesorului de comenzi etc.) alături de unele amănunte privind organizarea discurilor CP/M, structura și funcționarea nucleului CP/M ș.a., care contribuie la înțelegerea mai bună a modului în care lucrează sistemul și deci la utilizarea sa mai eficientă.

De asemenea, se prezintă sumar procedura de generare a unui nou sistem CP/M și anumite particularități de implementare mai importante pentru unele microcalculatoare românești.

2.1. Nucleul sistemului CP/M

2.1.1. Componentele nucleului : BIOS, BDOS, CCP

Sistemul de programe CP/M este organizat ierarhic, pe trei straturi succesive.

Stratul inferior, cel mai apropiat de mașina fizică, este constituit din componenta BIOS (Basic Input Output System).

BIOS este un ansamblu de subrutine de lucru cu dispozitivele periferice standard în CP/M: consola, discul, imprimanta, un dispozitiv de intrare și un dispozitiv de ieșire.

Stratul imediat superior este constituit din componenta BDOS (Basic Disk Operating System), a cărei sarcină principală o constituie gestiunea fișierelor disc.

BDOS este un ansamblu de subrutine de intrare-ieșire și de lucru cu fișiere disc, care fac apel, la rîndul lor, la subrutinele BIOS. În acest fel componenta BDOS este independentă de sistemul de intrare-ieșire al mașinii fizice.

Stratul exterior conține toate programele care folosesc funcții BDOS pentru operații de intrare-ieșire, inclusiv procesorul de comenzi consola CCP (Console Command Processor).

La încărcarea inițială a sistemului CP/M se aduc de pe disc în memorie, la adrese mari, componentele BIOS, BDOS și CCP, care formează împreună nucleul rezident al sistemului.

Modulele BIOS și BDOS trebuie să rămînă permanent în memorie, deoarece practic toate programele utilizate sub CP/M fac apel la funcții BDOS.

Modulul CCP asigură dialogul la consolă cu operatorul și este necesar în memorie numai între executarea a două programe succesive; în unele variante de sistem el rămîne însă permanent în memorie.

Memoria rămasă disponibilă după aducerea nucleului CP/M este utilizată pentru încărcarea şi executarea de programe diverse, cu excepţia primilor 256 de octeţi de memorie folosiţi tot de către nucleul CP/M.

Sistemul standard CP/M permite extinderea programelor peste componenta CCP din nucleu şi de aceea prevede reîncărcarea întregului nucleu sistem de pe disc la terminarea programelor mai mari, pentru readucerea programului CCP în memorie. Mai exact, un program executat sub CP/M este apelat de CCP ca o subrutină (printr-o instrucţiune CALL); terminarea unui program se poate face printr-o instrucţiune RET, dacă nu s-a alterat modulul CCP sau printr-un salt la adresa zero (sau prin apelarea funcţiei BDOS cu codul 0, care are acelaşi efect) pentru reîncărcarea nucleului CP/M

Majoritatea programelor CP/M se termină cu readucerea nucleului sistem în memorie, pentru mai multă siguranţă.

Desigur că este posibilă şi modificarea involuntară, prin erori de programare, a nucleului CP/M, dar atunci trebuie repetată procedura de încărcare iniţială.

Pentru a deosebi cele două situaţii de încărcare a nucleului sistem de pe disc se folosesc în CP/M termenii de *„încărcare la rece"* (cold start) pentru prima încărcare (cînd memoria RAM nu conţine nimic) şi de *„încărcare la cald"* (warm start) pentru încărcarea efectuată între programe (de către rutinele BIOS din memorie).

Structura ierarhică a sistemului CP/M este cea care asigură portabilitatea programelor (inclusiv a componentelor BDOS şi CCP) pe orice maşină care are un strat BIOS realizat conform convenţiilor CP/M.

Există însă anumite programe CP/M care nu pot fi scrise numai cu funcţii BDOS şi care folosesc direct instrucţiuni de intrare-ieşire ale procesorului, de obicei împreună cu funcţii BDOS (de exemplu programul de formatare discuri).

Modulul BDOS oferă, în afara funcţiilor de intrare-ieşire, o serie de alte funcţii sistem auxiliare, necesare programelor executate sub acest sistem. Numărul funcţiilor sistem oferite de CP/M prin BDOS este mai mic decît în alte sisteme de operare (sînt circa 40 de funcţii), iar dintre acestea mai puţin de jumătate sînt folosite frecvent.

2.1.2. Structura memoriei interne în CP/M

Memoria internă RAM într-o maşină CP/M este structurată în trei zone importante:

— în primii 256 octeţi de memorie se află o *zonă de lucru* a sistemului, utilizată şi de programele utilizator;

— imediat după zona sistem (de la adresa 100H) se află *zona TPA*, în care se încarcă şi se execută toate programele; limita superioară a acestei zone coincide cu începutul nucleului CP/M;

— *nucleul rezident* al sistemului CP/M se află la sfîrşitul memoriei.

Dimensiunea zonei TPA depinde pe de o parte de capacitatea totală a memoriei RAM şi pe de altă parte de mărimea nucleului CP/M rezident.

Memoria RAM necesară unei maşini CP/M este teoretic de 32 koct, dar de obicei are o capacitate între 56 şi 64 kocteţi.

Nucleul CP/M are o lungime variabilă de la o implementare la alta, cu valori uzuale între 7 şi 16 kocteţi, dintre care BDOS ocupă 3,5 kocteţi, iar CCP ocupă 2 kocteţi.

Dimensiunea zonei TPA este de obicei între 32 şi 48 kocteţi; într-o zonă TPA de 48 koct se pot executa toate programele existente, deoarece programele mai mari (DBASE, WordStar ş.a.) sînt segmentate, astfel încît să încapă în 48 koct de memorie.

Lungimea atît de diferită a nucleului CP/M rezultă din lungimea variabilă a modulului BIOS, determinată atît de secvenţele de cod cît şi de zonele de date necesare realizării funcţiilor de intrare-ieşire. De exemplu, utilizarea controlorului de ecran 8275 impune rezervarea unei memorii de ecran de circa 2 koct corespunzînd la 1920 caractere pe un ecran (cu 24 de linii a 80 de caractere fiecare).

Tot datorită acestor zone de date BIOS este posibil ca sfîrşitul zonei TPA să nu coincidă întotdeauna cu începutul modulului CCP.

Unele programe, cum ar fi DDT şi SID, se încarcă la sfîrşitul zonei TPA, reducînd în mod corespunzător lungimea acestei zone, cel puţin pe durata folosirii lor.

Singura zonă fixă din sistemul CP/M este zona de 256 de octeţi, dintre adresele 0 şi 100H, parţial folosită de sistem şi parţial de către celelalte programe.

Anumiţi octeţi din această zonă trebuie să aibă aceeaşi semnificaţie pentru orice sistem CP/M, dar ceilalţi octeţi nefolosiţi de sistem pot avea diferite utilizări în anumite variante de implementare sau pot rămîne nefolosiţi.

Prezentăm în continuare adresele fixe din *zona sistem CP/M:*

Octeţii 0, 1, 2 conţin o instrucţiune de salt către rutina de reiniţializare sistem din BIOS (rutina WBOOT).

Aceşti octeţi pot avea o dublă utilizare:

— un salt la adresa 0 poate fi folosit pentru terminarea unui program executat în zona TPA;

— adresa din locaţiile 1 şi 2, minus 3, reprezintă adresa de început a modulului BIOS pentru orice implementare şi poate fi folosită pentru apelarea rutinelor BIOS într-un mod independent de maşina şi de varianta CP/M utilizată.

Octetul 3 este octetul de intrare-ieşire (IOBYTE), care memorează asocierile între dispozitivele logice CP/M de I/E şi dispozitivele fizice de I/E. Acest octet poate fi folosit direct sau prin intermediul unor funcţii BDOS sau prin anumite programe utilitare (STAT de exemplu).

Octetul 4 conţine numărul discului implicit în biţii 0—3 şi codul utilizator curent în biţii 4—7 Acest octet conţine un număr binar egal cu zero pentru discul A, egal cu 1 pentru discul B ş.a.m.d. (am folosit în această frază cuvîntul „disc" cu sensul de „unitate de disc").

Octeţii 5, 6, 7. conţin o instrucţiune de salt către unicul punct de intrare al modulului BDOS. Şi aceste locaţii au o dublă utilizare:

— orice apel de funcţie BDOS generează un salt la adresa 5 (salt cu revenire, de tipul CALL);

— adresa din octeţii 6 şi 7 este utilizată ca adresă de sfîrşit a zonei TPA, de către multe programe şi permite verificarea depăşirii memoriei disponibile.

Octeţii de la 5CH la 7FH conţin *de obicei* un bloc descriptor de fişier (FCB = File Control Block) necesar pentru accesul la un fişier disc. Această zonă este completată de CCP şi folosită de multe programe de aplicaţii.

Octeţii de la 80H la FFH pot avea una dintre următoarele trei utilizări:

—pentru transmiterea liniei de comandă (a cozii comenzii, mai precis) de către CCP la programele apelate prin comenzi;

— ca zonă tampon implicită pentru acces la fişiere disc;

— ca stivă adresată de registrul SP pentru programele executate în zona TPA.

Imediat după încărcare, programul poate folosi zona de la adresa 80H pentru preluarea parametrilor comenzii, după care această zonă se foloseşte ca stivă sau ca zonă tampon. Registrul SP este încărcat de CCP cu adresa 100H, reprezentînd baza stivei implicite (stiva creşte spre adrese mici).

Trebuie reţinut că orice program 8080 sau Z80 are nevoie de o stivă pentru a lucra, dar dimensiunea maximă a acestei stive depinde de fiecare program; o lungime de 32 de cuvinte, deci de 64 octeţi, este în general suficientă pentru majoritatea programelor care nu folosesc recursivitate.

În concluzie, adresa 100H are o dublă semnificaţie în CP/M:

— este adresa de început a zonei TPA, care se extinde spre adrese mai mari;

— este adresa de bază a stivei implicite, care se extinde spre adrese mai mici.

2.2. Organizarea discurilor CP/M

2.2.1. Formatul fizic al discurilor CP/M

La un disc magnetic utilizat în sistemele de prelucrare a datelor deosebim un format fizic şi un format logic, specific sistemului de operare care foloseşte discul respectiv.

Formatul fizic se referă la numărul de piste şi de sectoare, la lungimea sectoarelor şi la modul de marcare a sectoarelor (modul de sectorizare), la informaţiile de control şi de sincronizare adăugate datelor înregistrate.

Formatul logic se referă la utilizarea pistelor şi sectoarelor de către sistemul de operare: de obicei un disc (un volum disc) are o etichetă de identificare a volumului şi o tabelă care reuneşte etichetele tuturor fişierelor de pe disc, după care urmează fişierele propriu-zise.

Fiecare sistem de operare foloseşte alte convenţii asupra formatului logic şi de aceea discurile cu acelaşi format fizic nu pot fi folosite direct (fără conversii de format) în alte sisteme de operare decît cel în care au fost înregistrate.

Discurile flexibile utilizate pe micro — şi minicalculatoarele româneşti au acelaşi format fizic, cunoscut şi sub numele de format IBM 3740 pentru discurile de simplă densitate şi IBM 3741 pentru discurile de dublă densitate. Acest format foloseşte o sectorizare soft, spre deosebire de alte discuri cu sectorizare hard (la care începutul fiecărui sector este marcat printr-o perforaţie pe disc).

Sectorizarea soft (prin program) înseamnă că delimitarea şi numerotarea sectoarelor disc se face prin înregistrarea unor informaţii de control cu ajutorul unui program de formatare (operaţia de formatare se mai numeşte şi iniţializare sau premarcare a suportului). În general, discurile flexibile sînt formatate de producător, purtînd şi o indicaţie a modului în care au fost formatate.

Multe sisteme de operare (cu excepţia notabilă a sistemului CP/M) necesită şi o iniţializare a formatului logic, în care se scrie eticheta de volum, se rezervă şi se iniţializează tabela de etichete a fişierelor (numită şi tabela directoare sau director) şi eventual alte informaţii.

De obicei există un singur program care face şi iniţializarea fizică şi iniţializarea logică, program specific fiecărui sistem de operare.

2.2.2. Formatul logic al discurilor CP/M

Un disc CP/M este un disc în care primele două piste (pistele 0 şi 1) conţin nucleul sistem, pista 2 conţine pe un număr de sectoare tabela directoare, după care urmează fişierele în format CP/M.

Uneori, primul sector din prima pistă conţine un program încărcător sau date pentru programul încărcător din memoria ROM, ceea ce permite o oarecare flexibilitate la încărcarea iniţială a sistemului de operare, inclusiv posibilitatea de a folosi diferite sisteme de operare pe o aceeaşi maşină.

Sistemul CP/M nu necesită o formatare logică, deoarece nu se folosesc etichete de volum, iar poziţia şi lungimea tabelei directoare sînt fixate în BIOS (de regulă primele 16 sau 32 de sectoare din pista 2).

Tabela directoare în CP/M este inițializată automat pe discurile noi sau formatate, deoarece un octet cu valoarea hexa E5, la început de etichetă de fișier, arată că poziția respectivă din tabelă este liberă, iar după formatare toate sectoarele discului sînt completate cu octetul E5H, repetat pe lungimea sectorului.

Ștergerea unui fișier de pe un disc CP/M se face prin introducerea valorii E5H în primul octet din eticheta fișierului respectiv, fără modificarea altor date din etichetă sau din fișier, ceea ce permite recuperarea fișierelor șterse din greșeală (cu condiția să nu fi fost create între timp alte fișiere pe discul respectiv).

Ordinea de memorare pe disc a componentelor nucleului CP/M este aceeași ca în memoria internă: CCP, BDOS și BIOS.

Tabela directoare reunește etichetele tuturor fișierelor de pe un disc și conține în principal numele și poziția pe disc pentru fiecare fișier în parte.

Lungimea și poziția tabelei de fișiere sînt de obicei aceleași pentru toate discurile CP/M cu același format fizic, astfel încît să se poată schimba fișiere între diferite mașini CP/M prin citire directă de pe disc, fără alte programe intermediare de conversie sau de adaptare. În practică această unificare s-a realizat pentru discurile de 8", de simplă densitate în toată lumea, iar pentru discurile de dublă densitate există compatibilitate între echipamentele fabricate la ICE (CUBZ, M216, M118b) și echipamentele fabricate la IEPER (TPD, JUNIOR).

Fișierele disc CP/M pot să nu ocupe zone continue pe disc, ele fiind formate dintr-o mulțime de blocuri disc situate teoretic oriunde pe volum.

Blocul (grupul) de 8 înregistrări CP/M are o lungime de 1 koct și reprezintă unitatea de alocare a spațiului pe disc în acest sistem.

Eticheta unui fișier cu lungime de maxim 16 koct conține numere (adresele) blocurilor disc ocupate de fișierul respectiv. Pentru fișierele mai mari de 16 k se folosesc mai multe etichete de lungime fixă (32 de octeți), toate cu același nume de fișier, dar cu alte adrese de blocuri.

Blocurile disc sînt numerotate de la 0, începînd din pista 2, astfel că directorul discului ocupă blocurile 0 și 1, iar fișierele ocupă blocurile de la 2 la 242 (pentru discuri de 8" în simplă densitate).

Separarea pistelor sistem de pistele care conțin fișiere în CP/M are avantajul că se pot rescrie pistele sistem fără să se afecteze fișierele existente. De exemplu, se poate face un disc sistem de la mașina X ca disc sistem pentru mașina Y, păstrînd fișierele existente.

Deoarece primele două piste rămîn oricum inutilizabile pentru fișiere, este bine ca orice disc CP/M să fie disc sistem pentru o mașină sau alta.

Imediat după copierea unor fișiere pe un disc gol, fișierele sînt continue, ocupînd blocuri succesive. După mai multe ștergeri și adău-

gări de fişiere pe un disc, se poate ajunge la situaţia în care unele fişiere mai mari sînt dispersate pe disc, ocupînd blocuri cu numere foarte diferite, eliberate prin ştergerea altor fişiere. Timpul de citire pentru asemenea fişiere dispersate va fi evident mai mare decît pentru cazul în care toate blocurile fişierului ar fi grupate într-o singură zonă disc.

Din punct de vedere al programării, plasarea pe disc a blocurilor unui fişier este absolut transparentă pentru utilizator, în sensul că acesta poate considera întotdeauna fişierul ca fiind o succesiune de blocuri.

De asemenea, invizibil pentru utilizatori este şi faptul că sectoarele unui bloc nu sînt sectoare fizic adiacente pe disc sau că unele blocuri se extind de pe o pistă pe alta.

Sistemul CP/M poate funcţiona cu mai multe unităţi de discuri de acelaşi tip sau de tipuri diferite: discuri flexibile de 8" cu diferite formatări, discuri flexibile de 5", discuri rigide cu diverse caracteristici. Sistemul de operare tratează în mod uniform toate unităţile de discuri, indiferent de tipul lor, în sensul că o unitate fizică sau o diviziune logică dintr-un disc este considerată ca un disc individual şi are un nume format dintr-o literă, urmată de caracterul ':'; astfel discul A: reprezintă unitatea 0, discul B: reprezintă unitatea 1 ş.a.m.d.

Numele unui disc poate apărea în comenzile CP/M singur sau în cadrul unui specificator de fişier.

2.3. Sistemul de intrări-ieşiri în CP/M

2.3.1. Dispozitive de intrare-ieşire şi fişiere

Spre deosebire de alte sisteme de operare, sistemul CP/M tratează diferit discurile magnetice faţă de celelalte dispozitive periferice, nemagnetice. Astfel, noţiunea de fişier se foloseşte în CP/M numai pentru discuri, nu şi pentru date pe alte suporturi.

Sistemul CP/M, ca şi alte sisteme, operează cu dispozitive periferice fizice şi cu dispozitive logice. În programele care folosesc funcţii BDOS se lucrează de fapt cu dispozitive logice, iar în comenzile de copiere, afişare ş.a. se pot folosi atît nume de dispozitive logice, cît şi nume de dispozitive fizice.

Asocierea dispozitivelor logice cu diferite dispozitive fizice depinde de fiecare sistem şi se poate modifica prin program sau de la consolă cu comanda STAT.

Prin schimbarea semnificaţiei concrete a unui dispozitiv logic este posibil ca un acelaşi program să poată lucra cu diferite dispozitive fizice, fără ca programul să sufere vreo modificare (de exemplu,

rezultatele se pot afişa la imprimantă sau la consolă, dacă nu este disponibilă o imprimantă).

În CP/M există următoarele patru *dispozitive logice*:
CON: (Console) consolă sistem
RDR: (Reader) dispozitiv de citire
PUN: (Punch) dispozitiv de perforare
LST: (List) imprimantă (dispozitiv de listare)

Fiecare dintre aceste dispozitive logice poate fi asociat cu unul dintre cele patru dispozitive fizice, permise pentru un dispozitiv logic. În octetul de I/E (de la adresa 3) se rezervă cîte 2 biţi pentru fiecare dispozitiv logic, după cum urmează:

0,1 CON: /2,3 RDR: /4,5 PUN: / 6,7 LST:

(bitul 0 este bitul cel mai puţin semnificativ, deci primul din dreapta)
Numele şi codificarea fiecărui *dispozitiv fizic* care poate fi substituit unui dispozitiv logic sînt date mai jos:

CON: TTY: (00), CRT: (01), BAT: (10), UC1: (11)
RDR: TTY: (00), PTR: (01), UR1: (10), UR2: (11)
PUN: TTY: (00), PTP: (01), UP1: (10), UP2: (11)
LST: TTY: (00), CRT: (01), LPT: (10), UL1: (11)

Semnificaţia asociată fiecăruia dintre aceste nume rezultă din modul în care este scris modulul BIOS.
Următoarele dispozitive sînt utilizate în majoritatea implementărilor CP/M cu aceeaşi semnificaţie:

TTY: şi CRT: consola sistem (de obicei cu ecran de afişare)
LPT: imprimanta sistemului

Numele celorlalte dispozitive fizice sugerează utilizarea lor posibilă sau lasă libertate totală celui care implementează un sistem CP/M:
BAT: (Batch) = Prelucrare serială în care consola pe intrare este înlocuită cu dispozitivul de citire (RDR:), iar consola pe ieşire este înlocuită cu dispozitivul de listare (LST:)

PTR: (Paper Tape Reader) = Cititor de bandă perforată
PTP: (Paper Tape Punch) = Perforator de bandă
UC1:, UR1:, UR2:, UP1:, UP2:, UL1: (User Console, Reader,...) = Dispozitive definite de utilizator.
Iată cîteva exemple de dispozitive de I/E utilizate pe microcalculatoare şi tratate cu dispozitive fizice în BIOS: lector de cartele, linii seriale de comunicaţie pe recepţie şi pe emisie ş.a.
Numele de dispozitive CP/M sînt terminate cu caracterul special ':' pentru a fi deosebite de numele unor fişiere disc în comenzile în care pot apărea ambele tipuri de nume (de exemplu, în PIP).

În anumite situaţii pot lipsi unul sau două dintre componentele specificatorului de fişier.

Discul suport poate lipsi atunci cînd fişierul se află pe discul implicit al sistemului.

Numele fişierului nu poate lipsi în general, decît la copiere de fişiere (programele PIP şi DIP), unde fişierul de ieşire primeşte automat numele fişierului de intrare.

Tipul fişierului poate lipsi în anumite comenzi care presupun implicit un anumit tip pentru fişierele de intrare sau care generează automat un tip predefinit pentru fişierele de ieşire (asambloare, compilatoare, linkeditoare ş.a.).

Fişierele disc pot avea în CP/M două *atribute:*

—protejat la scriere ($ R/O = Read Only) sau neprotejat ($ R/W);

—invizibil la DIR ($ SYS = System File) sau vizibil ($ DIR).

În mod implicit toate fişierele primesc la creare atributele $ R/W şi $ DIR, iar la copiere primesc atributele fişierului sursă; modificarea acestor atribute se poate face prin comanda STAT sau prin programare cu funcţii BDOS.

Operaţiile de I/E cu perifericele nemagnetice se realizează, atît în BIOS cît şi în BDOS, octet cu octet.

Operaţiile de I/E cu discul se desfăşoară înregistrare cu înregistrare, o înregistrare CP/M avînd lungimea de 128 octeţi, indiferent de lungimea sectorului disc (care poate fi multiplu de 128 octeţi)

Toate funcţiile BDOS şi BIOS de I/E implică aşteptarea terminării operaţiei, adică nu se revine în program pînă cînd nu se termină operaţia de I/E lansată. Pentru citire la consolă şi scriere la imprimantă este posibilă testarea sfîrşitului de operaţie prin funcţii separate de cele care transferă datele.

2.3.2. Subsistemul de intrări-ieşiri fizice BIOS

Modulul BIOS constă din 17 subrutine care trebuie să realizeze anumite funcţii şi să respecte anumite convenţii de transmitere a paraemtrilor.

Lungimile şi adresele acestor subrutine în cadrul modulului BIOS diferă de la o implementare la alta şi chiar de la o variantă de implementare la alta pentru aceeaşi maşină.

Pentru ca modulul BDOS să poată face apel întotdeauna la rutinele BIOS, indiferent de adresele acestora, se impune ca modulul BIOS să conţină la început un tabel de salturi către rutinele BIOS, într-o ordine prestabilită.

Ordinea instrucţiunilor de salt în acest tabel este aceeaşi pentru toate programele BIOS, asigurîndu-se în acest fel o interfaţă uniformă între BIOS şi BDOS.

Prezentăm mai jos tabelul cu care începe un modul BIOS, folosind numele de rutine din documentația originală CP/M, deși aceste nume nu au semnificație în afara programului BIOS:

```
JMP BOOT      ; Bootstrap = încărcare la rece
JMP WBOOT     ; Warm Bootstrap = încărcare la cald
JMP CONST     ; Console status = stare consolă
JMP CONIN     ; Console input = citire de la CON:
JMP CONOUT    ; Console output = afișare la CON:
JMP LIST      ; List output = ieșire la LST:
JMP PUNCH     ; Punch output = ieșire la PUN:
JMP READER    ; Reader input = citire de la RDR:
JMP HOME      ; Home position = poziție inițială pe disc
JMP SELDSK    ; Select disk = selecție unitate disc
JMP SETTRK    ; Set track = stabilire pistă
JMP SETSEC    ; Set sector = stabilire sector
JMP SETDMA    ; Set DMA address = stabilire adresă buffer
JMP READ      ; Read disk = citire sector disc
JMP WRITE     ; Write disk = scriere sector disc
JMP LISTST    ; List status = stare imprimantă
JMP SECTRAN   ; Sector translation = translatare sector
```

Rutina BOOT face toate inițializările necesare pentru interfețele de intrare-ieșire, pentru sistemul de întreruperi, inițializează registrul SP cu adresa 100H, scrie mesajul inițial al sistemului CP/M și completează anumiți octeți din zona sistem (octeții 0—7), iar la sfîrșit dă controlul programului CCP.

Este posibil ca unele inițializări pentru interfețele de I/E să fie făcute de către monitorul existent în memoria ROM, executat înainte de încărcarea sistemului.

Rutina WBOOT reface o parte din inițializările efectuate de rutina BOOT și încarcă nucleul sistem din pistele 0 și 1. În mare parte codul rutinelor BOOT și WBOOT este comun.

Programul BIOS conține atîtea rutine de intrare-ieșire cîte dispozitive fizice recunoaște și tratează sistemul respectiv, dar restul sistemului CP/M vede numai 5 rutine de citire-scriere pentru cele 4 dispozitive logice (pentru consolă sînt două rutine) și încă două rutine pentru citirea stării dispozitivelor "CON:" și "LST:".

Rutinele de scriere primesc octetul de date în registrul C, iar rutinele de citire depun octetul de date în registrul A.

Fiecare dintre cele 5 rutine menționate testează la început cei doi biți asociați dispozitivului logic, respectiv, din octetul de I/E (de la adresa 3) și selectează rutinele pentru dispozitivele fizice corespunzătoare.

Deoarece implementarea octetului de I/E este opțională în CP/M, de multe ori fiecare dispozitiv logic are un singur dispozitiv fizic asociat și nu se folosește octetul de I/E.

Celelalte rutine BIOS sînt destinate lucrului cu discurile magnetice și pot fi împărțite în două categorii:

— rutine de citire-scriere: READ, WRITE;

— rutine pregătitoare și auxiliare: HOME, SELDSK, SETTRK SETSEC, SETDMA, SECTRAN.

Rutinele de citire-scriere cu discul transferă între BIOS și BDOS cîte o înregistrare CP/M de 128 octeți, deoarece primele discuri folosite de sistemul CP/M aveau sectoare de această lungime; între disc și rutinele BIOS se transferă sectoare disc de lungime care poate fi diferită de 128, iar uneori chiar și o pistă întreagă. Aceste operații necesită precizarea următoarelor informații:

— numărul unității de disc;

— numărul pistei;

— numărul sectorului;

— adresa zonei tampon în care se citește sau de unde se scrie sectorul.

Coordonatele înregistrării trebuie comunicate înainte de citire sau scriere prin rutinele SELDSK (număr disc), SETTRK (număr pistă), SETSEC (număr sector) și SETDMA (adresă tampon). Primele 3 rutine primesc un număr de un octet în registrul C și depun acest număr într-un bloc de date din BIOS, folosit la execuția comenzii de citire sau de scriere. Rutina SETDMA primește adresa zonei tampon în perechea de registre BC.

Deoarece informațiile transmise prin aceste rutine se memorează în BIOS, nu este necesar să se facă apel la cele 4 rutine înainte de fiecare citire sau scriere, fiind suficientă transmiterea acelor parametri care se modifică de la operația anterioară la operația curentă.

În sistemele care operează cu discuri de densități diferite rutina SELDSK preia și sarcina de a recunoaște densitatea discului selectat, memorată în BIOS ca densitate curentă.

Rutinele READ și WRITE generează comenzile necesare interfeței de disc și așteaptă terminarea operației; ele fac degruparea și gruparea înregistrărilor din (în) sectoare fizice sau dintr-o pistă. La ieșirea din aceste rutine registrul A conține o valoare zero, dacă operația s-a terminat normal și o valoare nenulă dacă s-a produs o eroare de citire sau de scriere. Erorile de disc sînt semnalate de obicei de BDOS, deși rutinele BIOS pot furniza mai multe informații despre cauza erorii.

Rutina SECTRAN asigură transformarea unui număr logic de sector într-un număr fizic, folosind o tabelă de traducere, numită XLT. Renumerotarea sectoarelor se face de obicei din 6 în 6, pentru pistele de la 2 în sus, în vederea reducerii timpului de acces la disc.

Ideea acestei renumerotări a sectoarelor este aceea că după citirea sau scrierea unui sector se mai execută în general un număr de instrucţiuni şi deci se consumă un timp în care discul se roteşte şi se pierde începutul sectorului următor (fizic adiacent). Pentru a evita această situaţie se consideră ca sector logic următor un sector aflat la un interval oarecare faţă de sectorul curent; pentru discurile de 8" în simplă densitate acest interval se ia de 6 sectoare, deşi este o valoare cam mare.

În principiu, fiecare implementare a sistemului CP/M poate utiliza un alt tabel de translaţie XLT, dar compatibilitatea discurilor CP/M necesită utilizarea aceluiaşi interval pentru toate discurile cu acelaşi format fizic. Din acelaşi motiv nu se practică renumerotarea sectoarelor disc la formatare, deşi atît circuitul 8272 cît şi 8271 prevăd această posibilitate.

Rutinele BIOS folosesc mai multe *zone de date*, dintre care unele sînt comune tuturor implementărilor, iar altele sînt specifice fiecărei variante de BIOS.

Dintre zonele de date impuse, menţionăm blocul de parametri disc DPB, cîte unul pentru fiecare unitate fizică sau logică de discuri.

O zonă DPB are lungimea de 16 octeţi (extinsă uneori cu un octet de memorare a densităţii) şi conţine zone de lucru sau adresele unor zone: adresa tabelului de translaţie sectoare (XLT), adresa zonei tampon pentru o înregistrare din directorul discului (DIRBUF), adresa vectorului de alocare disc (ALV) şi a vectorului de verificare disc (CSV).

Dintre informaţiile memorate în DPB menţionăm: numărul de înregistrări pe pistă, lungime bloc (ca exponent al lui 2 pentru numărul de înregistrări), numărul de blocuri disc disponibile pentru fişiere (capacitate disc în kocteţi), numărul de etichete prevăzute în tabela directoare (număr maxim de fişiere pe un volum), numerele blocurilor ocupate de tabelul director, numărul de înregistrări din director folosite la verificarea discului.

Controlorul de disc 8272 necesită la rîndul său un tabel de parametri printre care: lungimea sectorului, număr de sectoare pe pistă, lungimea intervalului dintre sectoare etc.

Ambele categorii de parametri depind de dimensiunea şi de modul în care sînt formatate discurile flexibile; de asemenea, depinde de formatul fizic şi de conţinutul tabelului de translaţie a sectoarelor.

2.3.3. Subsistemul de fişiere BDOS

Componenta BDOS îndeplineşte în sistemul CP/M funcţia de gestiune a fişierelor disc, în sensul că asigură alocarea spaţiului pe disc pentru fişiere şi transformarea referirilor la fişiere în apeluri de rutine BIOS pentru citiri (scrieri) de sectoare disc.

Rutinele BDOS fac invizibile pentru programatori multe detalii legate de memorarea pe disc a fişierelor şi de acces la înregistrările fişierului.

Un fişier CP/M este o mulţime de înregistrări, identificată printr-un nume; înregistrările au întotdeauna lungimea de 128 octeţi. Mai exact, funcţiile BDOS de citire (scriere) asigură transferul între memorie şi fişier a unei înregistrări de 128 octeţi (un sector convenţional).

Din exterior (prin comenzi sistem) un fişier este complet identificat printr-un specificator de fişier; din programe un fişier este folosit printr-o zonă de descriere a fişierului FCB.

Un *specificator de fişier* CP/M are forma generală:

d: nnnnnnnn.ttt

d	este numele unităţii de disc pe care se află fişierul
nnnnnnnn	este numele fişierului (maxim 8 caractere)
ttt	este o extensie a numelui, care reprezintă de obicei tipul fişierului (maxim 3 caractere)

În cadrul numelui şi extensiei se pot utiliza litere, cifre şi caractere speciale (afişabile) cu următoarele excepţii:

〈 〉. , ; : = [ ]

În anumite situaţii se poate utiliza şi caracterul '?' într-un nume de fişier, pentru desemnarea unui grup de fişiere; în aceste cazuri semnul '?' poate fi înlocuit de orice caracter admis într-un nume de fişier.

Datorită structurii lor discontinue, fişierele CP/M se pot extinde dinamic prin adăugarea de noi înregistrări la sfîrşitul fişierului.

Ca şi în alte sisteme de operare, în CP/M sînt posibile două moduri de acces la fişiere:

— *acces secvenţial* la înregistrarea următoare din fişier;

— *acces direct*, pe baza numărului de înregistrare în fişier;

Înainte de a scrie sau de a citi dintr-un fişier, acest fişier trebuie deschis, iar la terminarea prelucrării unui fişier este necesară, în general, închiderea fişierului.

Există două operaţii de deschidere, distincte:

— deschidere fişier nou (pentru creare fişier);

— deschidere fişier existent (pentru exploatare sau actualizare).

După deschidere, referirile la fişier se fac prin intermediul unui descriptor de fişier, numit *FCB (File Control Block)*, folosindu-se adresa zonei FCB. Pentru fiecare fişier utilizat într-un program este necesar un bloc de control FCB.

Structura zonei FCB este aproape identică cu structura unei etichete de fişier din tabelul director.

În zona sistem, la adresa 5CH, este rezervată memorie pentru un bloc FCB, completat de către CCP cu numele a unul sau a două

fişiere ce apar în linia de comandă; pentru alte fişiere zonele FCB trebuie rezervate şi completate de utilizator.

Deschiderea unui fişier existent constă din căutarea în tabelul director a numelui de fişier dat în FCB şi din citirea în zona FCB a etichetei găsite (în caz că fişierul căutat există).

Eticheta de fişier conţine, pe lîngă numele şi tipul fişierului, numerele blocurilor disc ocupate de fişier, ceea ce permite regăsirea înregistrărilor din fişierul deschis.

Deschiderea unui fişier nou constă din scrierea conţinutului zonei FCB (numele şi tipul fişierului) în tabelul director al discului specificat în primul octet din FCB, ca etichetă a noului fişier.

Pe măsură ce se scriu noi înregistrări în fişierul deschis pentru creare, BDOS alocă alte blocuri disc libere pentru fişier şi trece numerele acestor blocuri în zona FCB.

Închiderea unui fişier constă din scrierea conţinutului zonei FCB pe disc, în eticheta fişierului asociat.

Deoarece completarea etichetei de pe disc cu numerele blocurilor alocate fişierului nu se face decît la închiderea fişierului (sau la depăşirea lungimii de 16 k, cînd se creează o nouă etichetă pentru acelaşi fişier), este esenţial ca un fişier creat sau extins prin adăugare de noi înregistrări să fie închis corect; în caz contrar eticheta de pe disc nu conţine lungimea reală a fişierului. Această situaţie nedorită apare la terminarea anormală a unui program, datorită unei erori hard sau soft (de exemplu, la editarea unui text).

Dacă s-au efectuat doar citiri dintr-un fişier existent, atunci eticheta de fişier nu suferă modificări şi se poate omite operaţia de închidere a fişierului (numai în CP/M, nu şi în alte sisteme de operare).

În realitate, conţinutul zonei FCB nu este chiar identic cu conţinutul etichetei de fişier nici imediat după deschidere; în continuare vom descrie exact cele două structuri de date.

Octetul 0 din etichetă conţine codul utilizatorului căruia îi aparţine fişierul (de obicei zero), iar octetul 0 din FCB conţine o codificare a discului suport. De reţinut că în FCB unitatea de disc este altfel specificată decît în octetul 4 din memorie:

codul 0 = discul implicit
codul 1 = discul A:
codul 2 = discul B:

Octeţii 1—11 sînt identici în FCB şi pe disc, deoarece aici se află numele fişierului (în octeţii 1—8, aliniat la stînga şi completat la dreapta cu blancuri) şi extensia numelui (în octeţii 9—11, aliniat la stînga şi completat cu blancuri).

Octetul 12 are aceeaşi semnificaţie în FCB şi în etichetă; el conţine numărul zonei disc de 16 koct din cadrul fişierului; pentru fişierele mai

mici de 16 koct, ca şi pentru prima zonă din fişierele mai mari, acest octet conţine 0. Acelaşi octet foloseşte şi pentru regăsirea, în ordine, a etichetelor referitoare la un acelaşi fişier mare, deoarece este posibil ca aceste etichete să fie memorate dispersat în tabelul director al volumului.

Octetul 13 este nefolosit în etichetă şi în FCB (are valoarea zero).

Octetul 14 este zero (nefolosit) în eticheta de pe disc, iar în FCB este utilizat de BDOS ca indicator de fişier deschis: înainte de deschidere trebuie să aibă valoarea zero, iar după deschidere primeşte o valoare diferită de zero.

Octetul 15 are aceeaşi semnificaţie pe disc şi în memorie; el conţine numărul de înregistrări din zona disc descrisă de eticheta respectivă (maximum 128 de înregistrări, corespunzînd la 16 blocuri de cîte 8 înregistrări fiecare).

Octeţii 16—31 din eticheta de fişier conţin numerele blocurilor alocate zonei de 16 koct descrisă de etichetă, în ordinea ocupării lor; zona FCB primeşte, după deschidere, acelaşi conţinut, care apoi poate fi modificat de BDOS pe măsură ce se alocă noi blocuri pentru fişier (adăugarea se face la sfîrşitul listei de blocuri, deci în ultima etichetă a unui fişier mare).

La trecerea de la o zonă disc la alta, rutinele BDOS de citire-scriere asigură citirea următoarei etichete (pentru zona disc următoare) din tabelul director în zona FCB.

Aşadar, conţinutul zonei FCB este, în general, modificat de către BDOS în cursul exploatării sau creării unui fişier disc.

O etichetă de fişier are lungimea de 32 de octeţi, dar zona FCB trebuie să aibă o lungime de 33 octeţi pentru acces secvenţial şi de 36 octeţi pentru acces direct.

Octetul 32 din FCB conţine numărul înregistrării următoare din cadrul zonei curente din fişier (între 0 şi 127); înainte de deschidere acest octet trebuie să fie zero, pentru a indica prima înregistrare din fişier imediat după deschidere.

Rutinele BDOS de citire sau scriere secvenţială actualizează acest octet, prin incrementare cu 1, după fiecare citire sau scriere.

Octeţii 33 şi 34 din FCB conţin numărul înregistrării din fişier care urmează a fi citită sau scrisă cu funcţiile BDOS de acces direct la înregistrare (octetul 33 este octetul inferior al unui număr binar pe 16 biţi). Acest număr este pus de programator şi folosit de BDOS.

Sistemul de fişiere BDOS detectează şi semnalează următoarele erori:

— încercare de scriere la o unitate de disc protejată la scriere *(„Disk R/O")*;

— încercare de scriere într-un fişier protejat la scriere *("File R/O")*;

— unitate de disc incorectă, selecţie imposibilă *("Select")*;

— eroare la citire sau scriere disc *("Bad Sector")*.

După mesajul „Bad Sector" se poate continua programul sespectiv prin apăsare pe clapa CR (Return), dar este posibil ca sectorul care a produs acest mesaj să nu fi fost bine citit sau scris.

După celelalte mesaje se poate reveni numai în sistem prin tastarea unui caracter (CR sau CTRL/C); de aceea trebuie prevăzute şi evitate asemenea erori (nu este posibilă tratarea lor prin program).

Cea mai frecventă situaţie care produce eroarea BDOS de disc protejat (Disk R/O) este cauzată de înlocuirea disketei într-o unitate, fără a semnala această schimbare sistemului prin tasta CTRL/C sau prin apel de funcţii BDOS (funcţia de selectare disc).

Această situaţie este proprie sistemului CP/M în care nu există pe fiecare volum disc un tabel de alocare a spaţiului de pe discul respectiv.

La iniţializarea şi reiniţializarea sistemului se citeşte tabelul director al discurilor din fiecare unitate şi se creează un vector de alocare a spaţiului pe disc ALV şi un vector de verificare disc CSV în zonele rezervate în BIOS.

Înainte de orice scriere pe disc se calculează o sumă de verificare pentru volumul din unitatea selectată şi se compară cu suma de control memorată pentru această unitate de disc; dacă ele diferă, atunci se refuză scrierea pe disc şi se emite mesajul de disc protejat.

Prin reselectarea unităţii de disc se reciteşte tabelul director al noului disc şi se reactualizează zonele ALV şi CSV; în caz contrar se pot face alocări greşite şi se pot distruge accidental fişiere de pe noul disc.

Componenta BDOS conţine, pe lîngă rutinele de lucru cu discuri, rutine de I/E pentru dispozitivele logice CP/M la nivel de octet, iar pentru consolă şi la nivel de linie. Aceste rutine diferă de rutinele BIOS corespunzătoare (pe care le apelează) prin aceea că interpretează o serie de caractere de control specifice sistemului CP/M.

Funcţia de citire caracter la consolă interpretează:

CTRL/C terminare program şi revenire în sistem
CTRL/P ecou la imprimantă (la LST:)
CTRL/S oprire afişare
CTRL/Q repornire afişare

Caracterele CTRL/S şi CTRL/Q sînt interpretate şi de către funcţia de afişare la consolă, care testează după fiecare caracter afişat starea consolei şi dacă s-a tastat unul dintre aceste caractere.

Funcţia de citire linie de la consolă mai interpretează în plus cîteva caractere de editare, folosite pentru corectarea caracterelor introduse greşit: şterge ultimul caracter introdus (BS şi DEL) şi şterge toată linia introdusă (CTRL/X şi CTRL/U).

2.4. Procesorul de comenzi consolă CCP

2.4.1. Comenzi consolă

Programul CCP este lansat după (re)inițializarea sistemului și la terminarea programelor tranzitorii.

Principalele funcții ale programului CCP, în ordinea realizării lor, sînt:

— caută pe discul sistem (pe unitatea A:) un fișier cu numele SUB. $ $ $ și, dacă îl găsește, atunci citește și interpretează comenzile din acest fișier;

— citește și analizează comenzi introduse de la consolă, completînd anumite zone sistem;

— dacă este o comandă rezidentă în CCP, atunci se execută comanda respectivă și se citește o nouă comandă;

— dacă nu este o comandă rezidentă în CCP atunci se caută în directorul discului specificat explicit sau pe discul implicit un fișier de tip COM avînd același nume cu numele comenzii; dacă se găsește, atunci CCP încarcă conținutul acestui fișier în zona TPA și execută un salt la adresa 100H pentru executarea programului încărcat;

— asigură schimbarea discului implicit dacă se citește un nume de disc care nu este urmat de un nume de fișier. ;

— reinițializează sistemul dacă se primește caracterul CTRL/C. Înainte de a citi o comandă de la consolă sau dintr-un fișier de comenzi, CCP afișează la consolă numele discului implicit urmat de caracterul '>' pentru ca operatorul să știe că este în dialog cu programul CCP și că acesta așteaptă introducerea unei comenzi.

O comandă CCP este un șir de maximum 128 caractere, terminat cu caracterul CR (Return) și are în general două părți:

— *numele comenzii*, care începe cu primul caracter diferit de blanc din linie și se termină la primul spațiu (blanc);

— *coada comenzii* conține toate caracterele dintre numele comenzii și terminatorul de linie CR, inclusiv blancurile.

Coada comenzii conține parametrii (datele) necesari executării comenzii și de obicei acești parametri (sau o parte dintre ei) reprezintă numele unor fișiere citite sau create prin comanda respectivă.

Numele comenzii este de obicei un nume de fișier, prin „nume" înțelegîndu-se un specificator de fișier.

Multe dintre comenzile CCP pot folosi ca parametri două tipuri de nume de fișiere:

— *nume neambiguu* („ufn" = unambiguous file name);

— *nume ambiguu* („afn"= ambiguous file name).

Un nume ambiguu poate conține caracterele speciale ‚?' și ‚*', înlocuibile prin orice caracter ('?') și, respectiv, prin orice șir de caractere ('*'), pentru a desemna un grup de fișiere.

Un nume neambiguu nu poate conţine caracterele '?' şi '*'; ei desemnează un singur fişier.

Numele de comenzi sînt nume neambigue de fişiere, iar unele comenzi nu admit nume ambigue (ex. TYPE, REN, PIP, DUMP ş.a.)

La introducerea de comenzi de la consolă se pot utiliza caracterele de control interpretate de CP/M, ca şi la introducerea de date pentru alte programe care folosesc funcţii sistem.

Pentru ştergerea ultimului caracter introdus se pot utiliza caracterele BS (Backspace) sau DEL (Delete, Rubout); pentru ştergerea liniei introduse în întregime (înainte de a introduce CR) se pot folosi caracterele CTRL/X sau CTRL/U. Caracterele BS şi X sînt destinate consolelor cu ecran, iar caracterele DEL şi U sînt destinate consolelor cu imprimare pe hîrtie.

În timpul executării unei comenzi se pot folosi caracterele universale de control a transmisei CTRL/S (X-OFF) pentru oprirea afişării şi CTRL/Q (X-ON) pentru repornirea afişării oprite cu CTRL/S; în CP/M se poate folosi tot CTRL/S şi pentru reluarea afişării.

Majoritatea comenzilor şi programelor afişează mesaje şi rezultate la consolă; pentru afişarea aceloraşi informaţii şi la imprimantă se poate folosi caracterul CTRL/P care are funcţia de a porni sau de a opri copierea la dispozitivul LST: a tuturor caracterelor afişate la dispozitivul CON:.

A se observa că o serie de caractere de control uzuale pot fi introduse printr-o singură tastă: CR (↑M) LF(↑J), BS (↑H), TAB (↑I) şi altele în funcţie de fiecare tastatură.

În cursul analizei unei linii de comandă programul CCP realizează următoarele:

— transferă coada comenzii (inclusiv spaţiile dintre numele şi coada comenzii) la adresa 81H, iar în octetul de la adresa 80H se introduce numărul de caractere din coada comenzii (fără CR şi LF, care nu fac parte din coada comenzii);

— extrage primele două subşiruri din coada comenzii (delimitate prin blancuri), le interpretează ca nume de fişiere şi completează *blocul FCB implicit* de la adresa 5CH după cum urmează:

oct. 00 = număr disc suport pentru primul fişier
oct. 01—08 = numele primului fişier (aliniat la stînga)
oct. 09—11 = tipul (extensia) pentru primul fişier
oct. 12—15 = zerouri binare
oct. 16 = număr disc suport pentru al doilea fişier
oct. 17—24 = numele celui de al doilea fişier
oct. 25—27 = extensia pentru al doilea fişier
oct. 28—31 = zerouri binare

Dacă coada comenzii conţine un singur subşir, atunci se completează numai primii 16 octeţi din FCB.

Zona de la adresa 5CH poate fi utilizată direct (dacă e un singur fişier parametru) sau după o mică prelucrare (dacă sînt două fişiere, se mută octeţii 16—31 în alt FCB) de către programele lansate de CCP, care sînt astfel scutite de analiza unui nume de fişier şi de completarea unui bloc FCB.

Trebuie menţionat că în cadrul acestei analize se expandează caracterele '*', recunoscute de CCP în caractere '?', recunoscute de BDOS. Caracterul '*' este înlocuit cu atîtea caractere cîte sînt necesare pentru completarea numelui sau extensiei. Exemplu:

WS*.*

va fi trecut în FCB ca

WS??????.???

care este diferit de

WS?.???

deoarece ultimele 5 caractere din acest nume vor fi blancuri.

În CP/M se implementează un mecanism simplu de protecţie a fişierelor aparţinînd unor utilizatori diferiţi, util în cazul discurilor rigide, de capacitate mare, pe care pot exista multe fişiere create de diferite persoane.

Fiecare fişier primeşte la crearea sa un *cod utilizator*, sub forma unui număr între 0 şi 15, înregistrat în eticheta fişierului. Pentru majoritatea comenzilor nu sînt vizibile decît fişierele avînd codul curent, cod memorat la adresa 4 şi stabilit sau modificat prin comanda USER.

Unele comenzi cu anumite opţiuni permit afişarea sau transferul de fişiere cu alte coduri decît codul curent: comanda D cu opţiunea /A, programul PIP cu opţiunea [Gn], programul POWER ş.a.

Trecerea unui fişier a cărui lungime nu depăşeşte zona TPA dintr-un cod în alt cod utilizator se poate face şi prin încărcare în memorie, urmată de salvare după schimbarea codului:

```
>USER 0
>DDT PIP.COM
—↑C
>USER 2
>SAVE 32 PIP.COM
```

Comenzile rezidente, interpretate de CCP, sînt fie comenzi des utilizate (DIR, ERA, TYPE, REN), care în felul acesta se execută mai rapid, fie comenzi scurte, care trebuie executate fără încărcarea altor programe (SAVE, USER).

Sintaxa comenzilor rezidente, folosind notaţiile „ufn" pentru nume neambigue de fişiere şi „afn" pentru nume ambigue, este urmă-

toarea (a se vedea şi paragraful 1.3 pentru descrierea acestor comenzi):

DIR	listare nume fişiere de pe discul implicit
DIR d:	listare fişiere de pe discul d: (d = A, B, C, ...)
DIR afn	listare fişiere desemnate prin numele afn
TYPE ufn	afişare conţinut fişier desemnat prin ufn
ERA afn	ştergere fişiere desemnate prin afn
REN ufn2=ufn1	schimbare nume fişier din ufn1 în ufn2
SAVE n ufn	scrie *n* pagini de 256 octeţi în fişierul ufn
USER m	stabileşte codul utilizator la valoarea *m*

Cunoaşterea modului în care lucrează sistemul CP/M şi programul CCP poate conduce la crearea de facilităţi care nu apar în documentaţia uzuală a sistemului; vom prezenta aici două exemple: posibilitatea de reluare a execuţiei unui program după revenirea în CCP şi execuţia automată a unei comenzi imediat după încărcarea sistemului.

La terminarea nedorită a unui program se revine în CCP; pentru continuarea programului respectiv ar trebui reluată execuţia programului, fără ca acesta să se reîncarce d pe disc. Această operaţie se poate realiza dacă avem pe discul respectiv un fişier cu conţinut nul, obţinut prin comanda SAVE astfel:

SAVE 0 RUN.COM

La comanda RUN (sau cu orice alt nume) programul CCP caută fişierul RUN.COM pe disc, încearcă să încarce conţinutul său, dar nu scrie nimic în memorie şi apoi execută un salt la adresa 100H; efectul acestei comenzi va fi deci cel de relansare a programului existent în zona TPA.

Executarea automată a unei comenzi iniţiale după iniţializarea sistemului CP/M se poate realiza prin introducerea acestei comenzi în zona tampon, în care se citesc comenzi de către CCP (situată la adresa CCP+8) folosind programele DDT şi SYSGEN sau CORSYS. Primul lucru pe care îl face programul CCP este să examineze lungimea (în octetul CCP + 7) şi conţinutul acestei zone tampon (dacă lungimea este nenulă); dacă există ceva în buffer, atunci se interpretează conţinutul lui drept comandă şi apoi se caută un fişier de comenzi sau se solicită comenzi la consolă.

Unele variante de sistem (TPD) folosesc însă acest buffer din CCP pentru unele secvenţe de iniţializare din BIOS, în care caz nu se poate modifica conţinutul său iniţial.

2.4.2. Fişiere de comenzi

Procesorul de comenzi CCP poate citi şi executa o secvenţă de comenzi dintr-un fişier, pentru a evita tastarea de către operator a fiecărei comenzi, mai ales cînd aceste comenzi conţin mai mulţi parametri.

Comenzile din fişier pot avea parametri variabili, de forma **$n**, care se înlocuiesc cu parametrii efectivi la apelarea fişierului de comenzi.

Programul utilitar SUBMIT este folosit pentru lansarea în execuţie a unui fişier de comenzi cu transmiterea parametrilor efectivi, dar programul CCP este cel care citeşte şi interpretează comenzile din fişier.

Un fişier de comenzi CP/M poate avea orice nume, dar trebuie să aibă extensia SUB; un astfel de fişier poate fi creat cu orice editor de texte. Programul SUBMIT citeşte fişierul de tip SUB, face substituţia parametrilor formali ($n) cu parametrii efectivi şi creează un fişier cu numele SUB. $$$.

Exemplu de fişier de comenzi, numit SUPERC. SUB, pentru compilare de programe C cu compilatorul Supersoft:

```
CC $1.C
C2 $1.COD
M80 $1 = $1.ASM
L80 $1, CRUNT2, FUNC, STDIO, ALLOC, C2RT, $1/N/E:
: CCSTAR
```

Exemplu de comandă SUBMIT pentru folosirea acestui fişier:

```
SUBMIT SUPERC TEST
```

Fişierul creat SUB. $$$ va avea iniţial următorul conţinut:

```
CC TEST.C
C2 TEST.COD
M80 TEST = TEST.ASM
L80 TEST, CRUNT2, FUNC, STDIO, ALLOC, C2RT,
TEST/N/E:CCSTAR
```

Înainte de a citi de la consolă o nouă comandă, monitorul de comenzi CCP, caută un fişier cu numele SUB.$$$ şi nu solicită comenzi de la consolă pînă cînd nu epuizează toate comenzile din acest fişier. Pe măsură ce mai citeşte şi execută o comandă din fişierul SUB. $$$, CCP şterge această comandă din fişier, iar la sfîrşit şterge şi numele fişierului de pe disc.

Această soluţie de tratare a fişierelor de comenzi în CP/M face ca timpul de interpretare a comenzilor din fişier să fie mai mare (la fiecare comandă se face o citire şi o scriere în fişierul SUB. $$$) dar este mai sigură: chiar dacă un program executat printr-o comandă din fişier produce reîncărcarea sistemului la terminarea sa, fişierul de comenzi rămîne pe disc şi va fi executat în continuare de la ultima linie netratată.

În CP/M nu există un limbaj de comenzi ca în alte sisteme de operare în sensul că nu există directive de executare condiţionată sau

repetată a unor comenzi și că singurele linii admise într-un fișier de comenzi sînt comenzi rezidente sau comenzi de apelare a programelor executabile.

Programul XSUB extinde posibilitățile fișierelor de comenzi prin aceea că programele executate prin comenzi din fișier își pot lua datele (propriile comenzi) din același fișier de comenzi și nu de la consolă. Exemplu de fișier de comenzi:

```
XSUB
DDT EX.COM
F200, 300,00
G0
SAVE 4 EX.COM
```

2.5. Implementarea unor sisteme CP/M

2.5.1. Generarea unui nou sistem CP/M

Implementarea sistemului CP/M-80 pe o nouă mașină presupune următoarele etape:

— scrierea și testarea unui modul BIOS pentru mașina respectivă;

— obținerea unui nucleu CP/M cu adrese corespunzătoare și combinarea modulului BIOS cu celelalte două componente ale nucleului într-un singur fișier;

— transferul nucleului CP/M în primele două piste ale unui disc, care devine disc sistem pentru noua mașină.

În practică pot apărea însă diferite probleme de implementare, dintre care menționăm depășirea limitei de 1 koct, care reprezintă lungimea maximă pentru modulul BIOS în sistemul de referință CP/M, și a limitei de 8 sectoare pe disc (în simplă densitate).

Pentru multe mașini care folosesc controlorul de disc 8272 și care nu beneficiază de rutine de I/E dintr-o memorie ROM (la care tot modulul BIOS se află în RAM) lungimea programului BIOS depășește cu mult dimensiunea prevăzută.

Pe discurile de simplă densitate extinderea nucleului sistem trebuie să se facă fără a modifica poziția tabelului de fișiere, pentru a nu pierde compatibilitatea cu alte sisteme; această restricție face aproape imposibilă realizarea de sisteme CP/M în simplă densitate pe asemenea microcalculatoare.

Pe discurile de dublă densitate problema măririi lungimii nucleului sistem se rezolvă automat prin faptul că în aceleași două piste sistem se pot memora cel puțin de două ori mai mulți octeți (la același diametru al discurilor).

În memoria internă sînt posibile două soluţii de extindere a nucleului rezident peste limita prevăzută:

— să se genereze sistem ca pentru o memorie RAM mai mică, iar zona rămasă disponibilă la sfîrşitul memoriei să fie utilizată pentru extensie BIOS;

— să se intercaleze extensia BIOS între sfîrşitul zonei TPA şi începutul modulului CCP.

Deoarece nucleul CP/M se poate afla la orice adrese, în funcţie de capacitatea memoriei RAM, s-ar părea că sînt necesare mai multe fişiere care să conţină acest nucleu asamblat cu diferite adrese de încărcare sau că sînt necesare programele BDOS şi CCP în forma sursă. În realitate, este suficient să existe un singur fişier numit de obicei CPMxx.COM (xx este capacitatea memoriei în kocteţi), al cărui conţinut poate fi translatat la alte adrese cu ajutorul programului MOVCPM (un relocator de nucleu CP/M).

Citirea în memorie a unui nucleu CP/M şi, respectiv, scrierea din memorie pe disc a acestui nucleu de la adresele stabilite de utilizator se fac cu ajutorul a două programe numite GETSYS şi PUTSYS (reunite uneori într-un singur program CORSYS), programe care folosesc funcţii BIOS de citire (scriere) pe disc.

Programul CORSYS, executat sub controlul DDT, permite şi efectuarea de corecţii minore într-un sistem, corecţii care nu necesită adăugarea unor secvenţe de instrucţiuni, ci numai modificarea unor date sau instrucţiuni.

Copierea pistelor sistem de pe un disc pe altul, deci multiplicarea discului sistem, se face cu programul SYSGEN.

2.5.2. Particularităţi de implementare CP/M

Vom prezenta aici unele particularităţi de implementare ale sistemului CP/M pe terminalele TPD şi JUNIOR, care reprezintă în acelaşi timp şi îmbunătăţiri aduse sistemului standard CP/M.

Perfecţionările aduse sistemului CP/M sînt de două tipuri: *creşterea performanţelor* sistemului prin reducerea timpului de lucru cu discul şi introducerea de *noi facilităţi de utilizare*.

Sistemul de operare CP/M este destinat în primul rînd dezvoltării de noi programe, ceea ce înseamnă că se execută un număr mare de programe relativ scurte (utilitare, procesoare de limbaje etc.) şi că o pondere importantă în timpul total de lucru o are timpul de citire-scriere pe disc.

Adăugînd la aceasta şi performanţele relativ modeste ale discurilor flexibile, folosite pe microcalculatoarele şi terminalele fabricate în ţară, rezultă că pentru creşterea productivităţii unui microsistem cu CP/M este importantă reducerea timpului de lucru cu discul.

Sistemul CP/M implementat pe terminalele TPD şi JUNIOR are o viteză de răspuns foarte bună, obţinută prin reducerea timpului de acces la disc atît la comutarea între programe, cît şi la încărcarea programelor de pe disc în memorie.

În sistemele CP/M utilizate pe TPD şi JUNIOR partea rezidentă, nealterabilă a sistemului include şi procesorul de comenzi CCP, precum şi alte zone de date utilizate de BIOS, cum sînt memoria de ecran şi zona buffer în care se citeşte o pistă disc. Ca urmare nu mai este necesară reîncărcarea sistemului între programe, deoarece nici un program nu poate modifica (decît accidental) vreo parte din nucleul CP/M.

Eliminarea operaţiei de reîncărcare sistem între programe conduce nu numai la reducerea timpului de lucru, dar creează şi noi facilităţi, dintre care cea mai importantă este posibilitatea de a scrie pe orice disc CP/M (de aceeaşi densitate) indiferent dacă este sau nu disc sistem pentru maşina respectivă. Sistemul standard CP/M nu permite scrierea pe discuri „străine" introduse în unitatea de discuri 0 (ele pot fi citite, dar nu pot fi scrise decît pe altă unitate de discuri).

O a doua soluţie de reducere a timpului de lucru cu discul, mai precis a timpului de citire de pe disc, constă în folosirea comenzii de citire a unei piste în locul comenzii de citire a unui sector în rutina BIOS de citire de pe disc. Toate circuitele de interfaţă pentru discuri flexibile dispun de o asemenea comandă, care permite citirea unei piste într-o singură rotaţie în loc de 4—5 rotaţii, dacă s-ar face citirea sector cu sector.

În sistemul standard CP/M rutinele BIOS de acces la disc citesc sau scriu sectoare fizice, a căror lungime variază între 128 şi 512 octeţi pentru discuri flexibile, în funcţie de densitatea aleasă la formatarea discurilor.

În sistemele CP/M utilizate pe TDP şi JUNIOR cu discuri de simplă densitate, rutina BIOS apelată pentru obţinerea unei înregistrări de pe disc citeşte cîte o pistă, dar citeşte numai atunci cînd sectorul cerut nu se află în pista deja citită în memorie.

Scrierea pe disc se face sector cu sector din mai multe motive: nu există de obicei o comandă de scriere a unei piste în întregime şi algoritmul de grupare a sectoarelor pentru scriere este relativ complex şi oarecum nesigur.

Trebuie observat însă că ponderea cea mai importantă o au operaţiile de citire de pe disc, majoritatea lor provenind din încărcarea în memorie a unor programe pentru a fi executate.

Preţul plătit pentru cîştigul de timp obţinut este reducerea zonei TPA disponibilă programelor cu 5,5 kocteţi, cît reprezintă lungimea procesorului de comenzi CCP (2 koct) plus dimensiunea zonei buffer în care se citeşte o pistă (3,5 koct în simplă densitate pentru o unitate de disc).

În cazul terminalelor TPD și JUNIOR care au o memorie RAM totală de 64 kocteți, zona TPA astfel redusă depășește limita de 48 kocteți suficientă pentru toate programele de sistem și de aplicații existente sub CP/M-80.

Aceeași soluție de mărire a vitezei de lucru este aplicată și la sistemele Felix M118, dar realizată printr-un program separat, executat după încărcarea sistemului.

O extindere a sistemului CP/M se referă la tratarea canalelor seriale de comunicație ca dispozitive fizice standard CP/M (cu numele UR1:, UR2: și, respectiv, UP1:, UP2:), dintre care unul dintre canale este implicit asociat dispozitivelor logice RDR: („Reader" = Cititor) și PUN: („Punch" = Perforator).

Practic, aceasta a însemnat introducerea în BIOS a rutinelor de emisie și de recepție pe liniile seriale, în mod asincron, cu protocol X-ON/X-OFF.

Pentru transmisia pe legătură serială se pot utiliza fie programul utilitar PIP, fie alte programe care fac apel la funcțiile BDOS cu codurile 3 și 4, fără ca programatorul să cunoască detaliile de lucru ale circuitelor de interfață (USART sau SIO) sau ale sistemului de întreruperi.

Prin recunoașterea și interpretarea unor caractere și ale unor secvențe de control ale ecranului în BIOS (poziționare pe ecran, ștergere ecran ș.a.) s-a realizat compatibilitatea în mod ecran între echipamentele TPD JUNIOR și M118, ceea ce permite utilizarea fără modificări a programelor care folosesc acest mod de afișare.

3. PROGRAME UTILITARE ÎN SISTEMUL CP/M

Sistemul CP/M este un sistem de operare conversațional, în care utilizatorul este atît programator, cît și operator la consolă.

Într-un asemenea sistem se creează, în cursul dezvoltării de programe, multe fișiere pe disc, unele cu caracter temporar, altele care trebuie să rămînă un timp mai îndelungat.

Spre deosebire de sistemele Felix-C, în care o serie de fișiere intermediare create de către programele de sistem rămîn necunoscute programatorilor și operatorilor (ca nume, dimensiune, disc suport), la microcalculatoare utilizatorul trebuie să țină evidența acestor fișiere și a volumelor disc pe care se află.

Utilizatorul CP/M atribuie de obicei nume acestor fișiere, precizează unitatea de disc pe care se află, șterge fișierele mai vechi sau devenite inutile atunci cînd nu mai este loc pe un volum, poate schimba numele unor fișiere, poate copia fișiere de pe un disc pe altul, poate modifica atributele unor fișiere ș.a.m.d. Precizăm că este vorba în principal de fișiere care conțin programe în diferite forme (sursă, obiect, executabil, biblioteci diverse), dar și despre fișiere de date.

Programele utilitare („Utilities") sînt destinate realizării unor operații oarecum auxiliare în raport cu programarea și cu exploatarea de aplicații, dar absolut necesare și destul de frecvente: inventariere de fișiere, copiere și concatenare de fișiere, copiere disc, ștergere fișier, schimbare nume fișier, creare de disc sistem, verificare discuri defecte ș.a.

Deoarece la microcalculatoare nu se folosesc de obicei cartele perforate, chiar programele sursă trebuie introduse pe disc în fișiere, fișiere care apoi se modifică pentru corectarea erorilor descoperite în aceste programe. Crearea și modificarea fișierelor sursă se face cu ajutorul editoarelor de texte, prezentate în capitolul următor.

Se pot deosebi două categorii de programe utilitare, concepute după principii opuse: utilitare mici (și multe) care îndeplinesc o singură funcție sau cîteva funcții interdependente și utilitare mari, care reunesc mai multe funcții diverse într-un program unic.

În CP/M, pe lîngă programele utilitare pentru creare și manipulare de fișiere, un loc important îl ocupă și programele care operează cu discul la nivel fizic pentru realizarea unor funcții cum ar fi: formatare discuri, verificare discuri, copiere discuri pistă cu pistă (orice format logic), vizualizarea și modificarea de sectoare disc, conversie dintr-un format logic în altul (de ex CP/M — SFDX) ș.a.

Sistemul CP/M se distinge între alte sisteme de operare prin numărul mare de programe utilitare, provenind din surse diverse, uneori realizînd aceleași funcții sau reunind funcții realizate de alte programe sub o formă diferită.

Mai trebuie observat că anumite programe utilitare depind de mașină sau de varianta de sistem folosită.

3.1. Programe pentru operații cu fișiere disc

Sînt reunite sub acest titlu programe utilitare ale căror funcții principale sînt:

— inventariere de volume disc (listare nume fișiere);
— copiere de fișiere;
— ștergere de fișiere;
— modificare atribute fișiere;
— listare conținut fișiere;
— comparare fișiere;

3.1.1. Programele de inventariere STAT, D, XDIR

Programul STAT (Status). Comanda STAT poate avea următoarele forme:

```
STAT
STAT d:
STAT afn
STAT d: afn
STAT d: = R/O
STAT ufn $atr
STAT VAL:
STAT DEV:
STAT 1d1 = pd1, 1d2 = pd2,...
```

Forma STAT afișează la consolă starea discurilor (R/W = read-write sau R/O = read only) și spațiul disponibil pe fiecare disc activ.

Forma STAT d: afișează starea și spațiul liber pe discul cu numele „d".

Forma STAT d: afn afişează informaţii despre fişierele desemnate prin numele ambiguu „d:afn": numărul de sectoare, numărul de octeţi, numărul zonelor de 16K, numele şi tipul.

Forma STAT d: = R/O protejează la scriere discul „d", stabilind pentru el atributul „read only".

Forma STAT ufn $atr stabileşte pentru fişierul cu numele „ufn" atributul „atr". Codurile atributelor permise sînt următoarele:

$R/O (read only) = protejat la scriere;
$R/W (read/write) = fişier accesibil la scriere;
$SYS (system) = fişier sistem, invizibil la DIR;
$DIR (directory) = fişier normal, vizibil la DIR;

Forma STAT VAL: afişează la consolă numele dispozitivelor fizice care se pot asocia fiecărui dispozitiv logic.

Forma STAT DEV: afişează atribuirile curente de dispozitive fizice la dispozitive logice.

Forma STAT 1d = pd atribuie dispozitivului logic cu numele „1d" dispozitivul fizic „pd", deci permite modificarea alocării dispozitivelor I/E.

Programul D (Disk Directory). Forma generală de apel este:

D [afn] [d: nume.tip] [/opţiuni]

Efectul comenzii D este afişarea numelor de fişiere desemnate prin numele ambiguu „afn" sau a tuturor numelor de fişiere, dacă lipseşte „afn", în ordine alfabetică şi grupate pe pagini de ecran, cu oprirea după fiecare pagină.

„d:nume.tip" poate fi inclus pentru a specifica un fişier în care să se înregistreze informaţiile afişate. În acest caz, trebuie inclus „afn", iar pentru fişierul de ieşire trebuie un specificator de fişier neambiguu.

Forma D afişează o listă sortată alfabetic a fişierelor de pe discul implicit.

Forma D VAL: afişează opţiunile programului D (ca şi la programul STAT).

Opţiuni:

/N dezactivare paginare;
/P ieşire către dispozitivul logic de listare (LST:);
/S afişare şi dimensiune fişiere;
/E afişare şi număr extensii şi atribute fişiere;
/T sortare după tipul fişierelor (implicit sortare alfabetică);
/A afişare informaţii despre toate codurile utilizator (0—16).

În mod normal se afişează numai fişierele de sub codul curent (de obicei 0).

Programul XDIR („Extended Directory"). Comanda XDIR poate avea două forme:

XDIR.
XDIR d:,

unde „d" este un nume de unitate disc (d = A,B,,C,D).

Efectul comenzii XDIR este afișarea la consolă a numelor tuturor fișierelor de pe discul implicit sau de pe discul specificat, ordonate alfabetic, împreună cu dimensiunea fiecărui fișier și cu spațiul liber rămas pe disc.

Programul XDIR este mai sensibil decît alte programe la anumite anomalii în tabela directoare a discului și nu afișează corect decît pentru discuri de simplă densitate; în aceste cazuri se va folosi unul din programele D sau STAT

3.1.2. Programele de copiere PIP, DIP

Programul PIP („Peripheral Interchange Program"). Comanda PIP poate fi utilizată sub două forme:

PIP
PIP dst = src1 [opt], src2 [opt],...

Forma PIP încarcă în memorie programul PIP și afișează un asterisc („*") pentru a semnala că este gata să accepte o linie de comandă; după executarea acestei comenzi se afișează din nou „*" și se poate introduce o nouă linie de comandă PIP. Această utilizare se recomandă atunci cînd se fac mai multe operații succesive cu ajutorul programului PIP, deoarece nu se mai reîncarcă în memorie de fiecare dată. Ieșirea din programul PIP astfel lansat se face introducînd caracterul ⟨CR⟩ ca răspuns la promptul „*" (o linie de comandă nulă).

O linie de comandă PIP are forma:

dst = src1[opt]), src2 [opt],.

A doua formă de apelare a programului PIP se recomandă atunci cînd el este folosit pentru o singură operație.

Notațile „dst" și „srcn" desemnează fie nume de fișiere disc (neambigue), fie nume de dispozitive periferice (logice sau fizice). Cu „opt" s-au notat una sau mai multe opțiuni (parametri) ce pot fi atașate fișierului sau dispozitivului sursă. Opțiunile se exprimă printr-o literă, eventual urmată de un număr întreg (în baza zece) și sînt încadrate între paranteze drepte.

Efectul comenzii PIP este de a copia datele citite din fișierele sursă sau de la dispozitivele sursă „srci", „src2",... în fișierul destinație (la dispozitivul destinație) „dst".

Dacă există deja un fișier cu numele „dst", atunci el este șters și înlocuit cu noul fișier creat de PIP (dacă operația s-a terminat normal).

Fișierele sursă și destinație se pot afla pe unități de discuri diferite.

Fișierele sursă sînt concatenate în ordinea în care ele apar în comanda PIP.

Sfîrșitul de fișier pentru fișierele ASCII este caracterul CTRL/Z, iar pentru celelalte fișiere este detectat de BDOS (prin contorizarea sectoarelor citite). Dacă fișierul destinație este un fișier ASCII, atunci programul PIP îi adaugă la sfîrșit un caracter CTRL/Z.

Este permisă și introducerea caracterului CTRL/Z de la consolă pentru fișierele sursă citite de la RDR:

Dacă destinația este un fișier disc, atunci se creează mai întîi un fișier destinație temporar (cu extensia „.$$$"), al cărui nume se schimbă (de către PIP) în numele indicat prin comandă numai la terminarea normală a operației de copiere (sau reunire de fișiere).

Operația de copiere realizată de PIP poate fi terminată forțat în orice moment prin apăsarea unei taste.

Este permis ca să nu se mai specifice numele fișierului destinație în care caz se preia numele fișierului sursă; de asemenea se poate indica numai numele fișierului destinație, luat implicit și ca nume pentru fișierul (fișierele) sursă. Exemple:

PIP B: = afn
PIP A: = B: afn
PIP ufn = B:
PIP A: ufn = B:

În toate exemplele de mai sus discurile sursă și destinație trebuie să fie diferite.

Pe lîngă numele de dispozitive logice și fizice CP/M programul PIP mai recunoaște și următoarele nume:

EOF: dispozitiv fictiv, care emite caracterul CTRL/Z, de sfîrșit de fișier ASCII.

PRN: dispozitiv de listare, similar cu LST:, de care diferă doar prin aceea că se expandează caracterele TAB din 8 în 8 poziții, se numerotează liniile și se inserează avans la pagină nouă după fiecare 60 de linii (și la început).

INP: dispozitiv de intrare, definit de utilizator, al cărui driver se conectează la PIP prin locația 103H. Programul PIP apelează adresa 103H pentru fiecare caracter și preia caracterul citit din locația 109H (fără bit de paritate).

OUT: dispozitiv de ieșire, definit de utilizator, al cărui driver se conectează la PIP prin locația 106H. Programul PIP apelează adresa 106H cu caracterul de afișat în registrul C, la fiecare caracter de transferat.

Programul PIP poate transfera programe în format hexazecimal, cu verificarea formatului corect (a sumei de control de la sfîrșitul fiecărei înregistrări), inclusiv în cazul în care sursa este un dispozitiv nemagnetic, iar destinația un fișier de tip „.HEX".

Opțiunile ce pot fi atașate fișierelor sau dispozitivelor sursă, fiind recunoscute de PIP, sînt următoarele:

B

Transfer pe blocuri: se citesc date din sursă în memorie pînă cînd se întîlnește un caracter CTRL/S și apoi se scrie acest bloc la destinație, după care se citește un nou bloc terminat cu CTRL/S ș.a.m.d. Acest mod este util la transferul de pe un suport secvențial (bandă, casetă magnetică) pe disc.

Dn

Se elimină (se şterg) caracterele ce depăşesc poziţia „n" dintr-o linie în cursul transferului. Se foloseşte la afişarea unor linii lungi la un dispozitiv de imprimare cu „n" caractere pe linie.

Se afişează în ecou la consolă toate caracterele transferate de la sursă la destinaţie.

F

Se elimină caracterele FF (Form Feed) în cursul copierii.

Gn

Fişierul sursă este preluat de sub codul utilizator cu numărul n (se poate folosi numai la fişierul sursă).

H

Transfer în format hexazecimal, verificîndu-se corectitudinea înregistrărilor şi aşteptîndu-se acţiuni de corectare de la consolă în caz de eroare.

I

Ignoră înregistrările care încep cu caracterele „:00" din formatul hexazecimal (opţiunea I implică şi opţiunea H).

L

Transformă literele mari în litere mici la transfer.

N

Se numerotează liniile transferate şi se adaugă numărul de linie la fiecare linie în parte (urmat de „:").

O

Transfer în format obiect (non-ASCII) de fişiere binare.

Pn

Se adaugă caractere FF (Form Feed) de avans la pagină nouă după fiecare „n" linii. Dacă n = 1 sau lipseşte atunci avansul de pagină se intercalează la fiecare 60 de linii.

Qs ↑ Z

Comandă terminarea transferului la întîlnirea şirului de caractere „s", terminat prin CTRL/Z (se transferă şi „s").

Ss ↑ Z

Comandă reluarea transferului de la întîlnirea şirului de caractere „s", terminat cu CTRL/Z. Parametrii S şi Q pot fi utilizaţi pentru extragerea anumitor secţiuni dintr-un fişier.

Tn

Se expandează caracterele TAB din fişierul copiat, prin salturi din *n* în *n* poziţii.

U

Transformă literele mici în litere mari la copiere.

V

Verificarea copierii corecte prin recitirea sursei şi compararea cu destinaţia (destinaţia trebuie să fie un fişier pe disc).

Z
Pune zero în bitul de paritate al fiecărui caracter transferat.

Opţiunile pot fi scrise cu litere mari sau cu litere mici.

Exemple de utilizare a opţiunilor PIP:

PIP X.ASM = B: [V]

PIP LST: = X.ASM[NT8U]

Programul DIP („Disk Interchange Program"). DIP este un program de copiere asemănător cu PIP, dar care transferă numai fişiere disc şi care are o listă de opţiuni diferită. DIP este destinat pentru copiere selectivă de fişiere de pe o dischetă pe alta folosind o singură unitate de discuri.

Forme de apel:

DIP

DIP afn2 = afn1

Prima formă permite execuţia unui şir de comenzi introduse individual; înainte de a tipări promptul '*', se afişează un mesaj explicativ. Cea de a doua formă execută imediat comanda specificată după care controlul este dat sistemului.

Opţiunile se introduc între paranteze drepte şi pot fi:

P
Pauză pentru schimbarea discurilor sursă şi destinaţie (opţiune implicită dacă se copiază pe aceeaşi unitate).

Q
Cere confirmarea operaţiei pentru fiecare fişier.

V
Verificarea transferului prin citirea fişierului creat.

În cazul în care specificatorii sursă şi destinaţie selectează aceeaşi unitate de disc, opţiunea P este automat activă.

Programul DIP permite copierea unor fişiere de orice lungime prin segmentarea lor automată, precum şi utilizarea eficientă a memoriei disponibile în cazul copierii mai multor fişiere mici.

DIP reciteşte automat directorul discului introdus în unitate şi verifică reintroducerea corectă a discurilor sursă şi destinaţie.

Pe parcursul copierii se foloseşte un fişier temporar cu numele DIP.$$$, al cărui nume se schimbă în numele fişierului destinaţie, dacă operaţia se termină cu succes.

Numele fişierului destinaţie poate lipsi, fiind preluate automat numele şi extensia fişierului sursă. Exemplu:

DIP A: = CPM.DOC[V]

3.1.3. Programe utilitare diverse

Programul DUMP. Comanda DUMP are forma următoare:

DUMP ufn

Efectul ei este de afişare la consolă a conţinutului fişierului desemnat de „ufn", în hexazecimal, împreună cu adresele de octet în cadrul fişierului (se afişează cîte 16 octeţi pe o linie). Se foloseşte pentru fişiere de tipul COM. Afişarea prin „DUMP" poate fi întreruptă cu tasta DEL.

Reamintim că pentru aceeaşi operaţie poate fi folosit şi un program depanator (DDT, SID)

Programul COMPARE. Programul COMPARE compară două fişiere, octet cu octet şi afişează sectoarele din cele două fişiere unde apare prima diferenţă. Sintaxa comenzii este următoarea:

COMPARE ufn1 ufn2

Fişierele comparate pot fi de orice tip, cu orice conţinut.

Programul UNERA (Unerase). Se foloseşte pentru recuperarea unui fişier şters din greşeală.
Comanda UNERA are forma

UNERA ufn

Programul ERAQ (Erase with querry). Se foloseşte pentru ştergerea selectivă de fişiere cu confirmare din partea operatorului. Comanda ERAQ are forma

ERAQ afn

3.2. Programe de lucru cu discul la nivel fizic

3.2.1. Programe de formatare-verificare şi copiere fizică

Programul SYSGEN („System Generator"). Programul SYSGEN se foloseşte pentru copierea nucleului CP/M de pe un disc sistem pe un alt disc, deci pentru crearea unui nou disc sistem. Comanda SYSGEN are forma:

SYSGEN

şi declanşează un dialog cu operatorul, prin care se precizează parametrii de generare.

Mai întîi se cere numele discului sursă pe care se află sistemul (A:, B:, C:, D:); dacă nucleul ce urmează a fi scris se află deja în memorie ca urmare a unei comenzi MOVCPM, atunci se introduce doar ⟨CR⟩, apoi se cere numele discului pe care se va înregistra noul sistem şi acest mesaj este repetat pentru a permite generarea mai multor discuri sistem cu o singură comandă. Ieşirea din acest ciclu se face tot cu caracterul ⟨CR⟩.

Programul SYSGEN nu scrie decît pistele 0 şi 1 de pe discul destinaţie (deci nucleul CP/M), fără să afecteze celelalte piste.

Comanda SYSGEN este necesară chiar dacă se doreşte un nou disc identic cu discul sistem de la care se pleacă, deoarece pistele 0 şi 1 nu pot fi copiate cu PIP sau cu DIP.

Programe de formatare (FORMAT, IDISK ş.a.). Există mai multe programe de formatare a discurilor, cu diferite nume. Programul de formatare depinde de echipamentul utilizat şi de aceea trebuie ca el să corespundă cu calculatorul şi cu versiunea sistemului de operare.

Unele dintre aceste programe permit diferite formatări pe baza unor opţiuni ale utilizatorului: formatare în simplă densitate sau formatare în dublă densitate cu sectoare de 256 sau de 512 octeţi.

Programe de verificare (DOCTOR, VDISK ş.a.). Verificarea se face atunci cînd se constată erori la copiere de fişiere sau la copierea pistelor sistem. O bună verificare a calităţii suportului se face numai prin scriere urmată de citire, ceea ce durează mult şi poate distruge informaţii utile de pe discul cu piste defecte: de aceea uneori verificarea se face numai prin citire (de exemplu , în programul POWER).

În urma găsirii unor sectoare defecte, se afişează numerele sectoarelor respective, iar uneori (POWER ş.a.) se creează automat un fişier cu un nume special care acoperă blocurile defecte.

Uneori simpla reformatare a discului conduce la eliminarea defectelor constatate; înainte de formatare se vor copia fişierele bune pe un alt disc.

Folosind programele DU sau POWER se pot recupera părţi din fişiere care au blocuri defecte.

Programe de copiere fizică. Copierea pistă cu pistă a unui disc în întregime poate fi mai rapidă decît copierea tuturor fişierelor şi se poate utiliza pentru discuri care nu au format logic CP/M; acest mod de copiere presupune însă că nu există defecte sau erori nici pe discul sursă nici pe discul destinaţie.

Există mai multe programe de copiere cu nume diferite; unele pot opera cu o singură unitate de discuri, iar altele necesită două unităţi.

Programul UDISK. Acest program, folosit pe terminalele TPD şi JUNIOR, reuneşte funcţiile de formatare, verificare şi de copiere fizică; este uşor de folosit, deoarece afişează opţiunile posibile şi solicită codul operaţiei.

Programe de conversie format logic. Aceste programe se folosesc pentru citirea, scrierea sau inventarierea de discuri străine sub sistemul CP/M.

Programul Q asigură transferul în ambele sensuri între CP/M şi SFDX-18 precum şi inventarierea de discuri SFDX; el necesită două unităţi de discuri.

3.2.2. Programul editor de disc DU (Disk Utility)

Programul DU permite vizualizarea şi modificarea conţinutului unui disc flexibil la nivel fizic, de pistă şi sector.

DU este destinat în primul rînd pentru refacerea sau corectarea unor anomalii în directorul unui disc CP/M, dar poate fi folosit şi pentru

discuri cu alt format logic, dar cu format fizic IBM 3740 (77 de piste a cîte 26 sectoare de 128 octeți fiecare).

Programul DU folosește la afișare următoarele prescurtări:

T (Track): Pistă (00—4DH)
S (Sector): Sector logic
PS (Physical Sector): Sector fizic (00—1AH)
G (Group): Grup de 8 sectoare logic succesive (bloc CP/M)

Numerotarea sectoarelor logice este identică cu numerotarea sectoarelor fizice în pistele 0,1 și folosește un factor de întrețesere egal cu 6 pentru pistele 2—76.

Grupul 0 începe în pista 2, sectorul 1 și conține sectoarele logice 1—8, deci sectoarele fizice 1, 7, 13, 19, 25, 5, 11, 17.

Toate constantele numerice introduse prin comenzi DU sau afișate de DU sînt în hexazecimal.

Comenzile DU pot fi introduse cîte una pe o linie, fiecare terminată cu ⟨CR⟩, sau mai multe comenzi eparate prin "; " într-o aceeași linie terminată cu ⟨CR⟩.

În comanda de lansare a programului DU se poate utiliza ca parametru o secvență de comenzi DU separate prin „; ". Exemplu:

DU G0; D; G2; = END⟨0D⟩ ⟨0A⟩ ⟨1A⟩; D

Lista comezilor DU:

?

Afișare sumar de comenzi (Help).

Tnn

Poziționare pe pista cu numărul „nn", fără citire de pe disc.

Snn

Poziționare pe sectorul logic "nn" și citire sector.

Gnn

Poziționare pe primul sector din grupul „nn" și citire sector.

G

Afișarea poziției curente: Pistă, Sector, Grup.

\+ sau + n

Poziționare înainte peste unu sau peste "n" sectoare logice

— sau —n

Poziționare înapoi cu unul sau cu „n" sectoare logice.

V sau Vnn

Vizualizarea a unul sau a „nn" sectoare în ASCII.

Fnume

Căutarea fișierului cu numele „nume" și poziționare pe sectorul din director care conține acest nume, cu afișarea sectorului.

= sir

Căutarea pe disc a unui șir de caractere „sir" începînd cu sectorul curent. În șirul căutat pot fi inserate constante hexa între parantezele unghiulare „⟨" și „⟩". Exemplu: = OUT ⟨09⟩ ⟨81⟩.

Ld
Stabilirea unitate de disc „d" ca disc curent (d = A,B,..).

L
Reciteşte directorul discului din unitatea curentă (Login).

D sau Ds, f
Afişare sector curent în hexazecimal şi ASCII (Dump).

A sau As, f
Afişare sector curent în ASCII.

H sau Hs, f
Afişare sector curent în hexazecimal. Adresele de start „s" şi final „f" sînt adrese relative în cadrul sectorului curent între care se face afişarea.

CHa, bl, b2,..
Modificare hexazecimal la adresa „a" din sectorul curent prin introducerea octeţilor „b1, b2,.." (Change Hexa).

CHa1 — a2,b
Copiere repetată a octetului „b" între adresele „a1" şi „a2" din sectorul curent.

CAa, sir
Modificare ASCII la adresa „a" din sectorul curent prin introducerea şirului de caractere „sir".

CAa1 — a2, b
Copierea repetată a caracterului ASCII „b" între adresele „a1" şi „a2" din sectorul curent.

Observaţie.
Comenzile CH şi CA modifică numai imaginea sectorului curent din memorie; pentru a se modifica şi pe disc se foloseşte comanda „W".

R
Citire sector curent în memorie de pe disc (Read).

W
Scriere sector curent din memorie pe disc (Write).

<
Salvare sector curent într-o zonă de memorie.

>
Readucere sector salvat ca sector curent.

/nn
Repetă comanda sau şirul de comenzi anterior de „nn" ori. Numărul „nn" este exprimat în zecimal şi nu poate depăşi 254.

M
Afişare numere grupuri ocupate de un fişier (Map).

Mn
Afişare nume fişier care conţine grupul numărul „n".

P
Activare sau dezactivare ecou la imprimantă (Print).

Z sau Znn
Pauză de o secundă sau de „nn" zecimi de secundă.

Exemple de utilizare a comenzilor DU: Ştergerea directorului discului din unitatea B: cu valoarea E5H se poate face cu comenzile:

LB comutare pe unitatea B
G0 poziţionare pe început de director
CH0—7F, E5 scrie 128 de octeţi E5 în sectorul curent
< salvare sector (cu E5)
>; W; +; /16 repetă de 16 ori: readucere 128 octeţi cu E5, scriere pe disc, avans la sectorul următor.

Acelaşi efect se poate obţine prin numai două linii:

LB; G0; CH0—7F, E5; <
>; W; +; /16

Observaţie. Comanda CP/M ERA B: *.* nu are acelaşi efect, deoarece se pune E5 numai în primul octet din etichetele de fişier care aveau zero (sau codul utilizatorului) în acest octet. În acest fel este posibil ca diskete utilizate anterior sub sistemul de operare SFDX să nu poată fi şterse corect (total) prin comanda ERA. Numai formatarea unei diskete completează cu E5 toţi octeţii din toate sectoarele.

3.3. Programe utilitare integrate

Există o serie de programe care integrează funcţii realizate prin diverse alte comenzi şi programe şi care prezintă o interfaţă proprie cu utilizatorul, care se substituie interfeţei asigurate de CCP.

3.3.1. Programul POWER

Programul POWER reuneşte o serie de funcţii realizate de comenzile rezidente şi de programele STAT, PIP, DU, DDT ş.a., într-o formă unitară, cu multe facilităţi suplimentare.

Principalele tipuri de operaţii realizate de POWER sînt:

— inventariere fişiere şi volume;
— operaţii asupra fişierelor: schimbare nume, ştergere, copiere cu verificare (dar nu pe o singură unitate), afişare conţinut fişier;
— executarea unui program, fără ieşire din POWER;
— salvare din memorie pe disc;
— afişare (modificare) conţinut memorie şi căutare în memorie;
— citire (scriere) de sectoare şi blocuri disc;
— testare disc sector cu sector;
— afişare parametri disc;
— modificare cod utilizator.

Programul POWER începe prin afişarea unui caracter prompter („=”), de solicitare a unei comenzi.

La cerere (prin caracterul „?") POWER poate afişa numele comenzilor disponibile, dar nu şi sintaxa acestor comenzi.

Pentru comenzile care operează cu fişiere, POWER afişează pe ecran numele fişierelor de pe discul implicit sau de pe discul specificat. însoţite de cîte un număr de ordine; utilizatorul poate selecta fişierele ce urmează a fi prelucrate de comandă printr-o listă de numere de fişiere, ca în exemplele următoare:

1 5 9 (fişierele 1,5 şi 9)
1—7 (fişierele 1 pînă la 7)
12— (fişierele de la 12 pînă la sfîrşit)

Este posibilă specificarea într-o comandă a unui grup de fişiere printr-un nume ambiguu (folosind caracterele „?" şi „*"); urmînd ca apoi să se facă selecţia prin număr în cadrul acestui grup.

O convenţie proprie POWER este secvenţa de 3 asteriscuri,***. care semnifică toate fişierele şi numai este urmată de o cerere de selecţie prin numere de fişier (spre deosebire de *.*).

Fişierele protejate la scriere „R/O" vor fi ignorate la executarea unor anumite funcţii (REN, ERA...).

După comenzile care prelucrează fişiere sau discuri se poate introduce numele unui disc, în cazul în care comanda nu trebue să lucreze pe unitatea curentă.

Apăsînd CTRL/C se întrerupe execuţia unei comenzi, iar POWER se prezintă din nou cu caracterul prompter.

Dacă se schimbă discurile în timp ce se lucrează cu POWER, este necesară apăsarea lui CTRL/C, pentru a evita protejarea unităţilor de disc de către BDOS. În plus CTRL/C întrerupe orice comandă.

În locul unei comenzi sau împreună cu o comandă imperativă se pot introduce următoarele opţiuni de lucru, încadrate între paranteze drepte (cîteva dintre opţiuni acţionează ca nişte comutatoare activînd sau dezactivînd funcţia respectivă):

P tipărire la imprimantă a ceea ce se afişează pe ecran
R se cere confirmare înainte de execuţia oricărei comenzi
i este numărul de coloane utilizate pentru afişarea directorului (1—9)
V verificare la scriere (citire după scriere)
S afişarea fişierelor cu atributul SYS
U afişarea directorului grupat pe coduri utilizator

Comenzi acceptate de POWER:

Comenzi generale

?

Afişare comenzi disponibile.

. d:

Selectare unitate de disc cu numele d (A,B,..).

EXIT

Terminare POWER şi reîntoarcere în sistemul de operare.

USER i

Comutare pe codul utilizator „i".

XUSER i

Stabilește „i" drept cod utilizator pentru fișierul destinație.

LOG

Afișare opțiuni disponibile (comutabile), dimensiuni POWER și zona TPA.

SPEED i

Comandă viteza cursorului.

SPEED 0 repede
SPEED i încet (i = 0...9)

În timpul oricărei afișări sînt active următoarele taste:

„0" la „9" — control viteză cursor
„spațiu" — stop

După stop, sînt recunoscute:

„spațiu" — stop după o linie;
<CR> — stop după o pagină;
<LF> — stop după o pagină;
alt caracter — continuare afișare

UR i

Comenzi utilizator (i = 1...4). Pot fi apelate numai dacă sînt introduse în următoarele locații de memorie din POWER:

UR1 140...147 H
UR2 148..14FH
UR3 150..157H
UR4 158..15FH

Comenzi de lucru cu tabela directoare

DIR [opt]

Afișarea directorului discului curent sau a discului specificat, dacă după comandă urmează un nume de disc, de exemplu: DIR B: [U].

Opțiuni:

U afișare director grupat pe coduri utilizator:

X afișare tabele directoare pentru unitățile de disc selectate de la ultimul CTRL/C;

i i = 1...9; numărul de coloane folosite pentru afișarea tabelului director.

Pot fi introduși mai mulți parametri deodată, fără spații între ei: DIR [UX].

Caracterele afișate în director au următoarea semnificație:

* fișierul are atributul R/O (protejat la scriere);
( ) fișierul are atributul SYS (invizibil);
> fișierul sursă într-o operație de copiere;
< fișierul este rezultatul unei operații de copiere.

REN

Redenumirea fişierelor selectate. Pentru fiecare fişier se cere noul nume. Introducerea unui caracter „*" în zona „nume" sau „ext" lasă vechiul nume respectiv extensia neschimbată.

ERA

Ştergerea fişierelor selectate. Dacă opţiunea R este activată, înainte de ştergerea fiecărui fişier se mai cere o ultimă confirmare.

SORT i

Sortarea tabelului director în funcţie de parametrul „i":

i = o nesortat;
i = 1 după numele fişierelor;
i = 2 după numele fişierelor, fişierele „SYS" la urmă;
i = 3 după extensia numelui fişierelor;
i = 4 după extensia numelui , fişierele „SYS" la urmă.

SETRO

Stabileşte atributul R/O pentru fişierele selectate. Vor fi afişate pentru selecţie numai fişierele cu atributul R/W.

SETWR

Stabileşte atributul R/W pentru fişierele selectate. Vor fi afişate pentru selecţie numai fişierele cu atributul R/O.

SETDIR

Stabileşte atributul DIR pentru fişierele selectate.

SETSYS

Stabileşte atributul SYS pentru fişierele selectate.

SET [+n] [—n]

Pune pe 1 (—n) sau pe zero (+n) bitul 7 din octetul „n" al numelui fişierului. SET fără parametrii afişează biţii 7 setaţi. În locul numărului de bit „n" se pot utiliza şi literele:

R: primul caracter al extensiei numelui fişierului (R/O)
S: al doilea caracter al extensiei numelui (SYS)
X: al treilea caracter din extensie

Exemple:

SET [—1] setează bitul 7 din primul octet al numelui.

SET [+R] resetează bitul 7 din primul octet al extensiei numelui fişierului (ştergerea atributului „R/O").

SIZE

Afişarea necesarului de memorie disc pentru fişierele selectate. Vor fi afişate: sectoare ocupate, sectoare libere, număr de kiloocteţi, total ocupare.

RECLAIM

Recuperare fişiere şterse. Vor fi afişate unul după altul fişierele şterse care există în directorul discului; se răspunde cu „Y" dacă se doreşte recuperarea fişierului şters.

GROUP

Se afişează pentru fiecare fişier selectat numărul blocurilor ocupate şi numărul de etichete din director.

CHECK

Afişarea sumei de control a zonei de date pentru fişierele selecţionate.

Comenzi de lucru cu fişiere

TYPE afişare caractere ASCII (fără sectorizare);
TYPEX afişare hexazecimală şi ASCII;
TYPEH afişare hexazecimală;
TYPEA afişare ASCII.

Caracterul ↑K forţează saltul la următorul fişier

COPY [Q]

Copiere fişiere selectate pe un alt disc. Numele discului destinaţie va fi introdus după mesajul „destination drive". Opţiunea „Q" permite schimbarea numelui la copiere.

Opţiuni pentru COPY:

C Se verifică dacă fişierul există deja pe discul destinaţie.
Dacă există, atunci se poate răspunde prin:
- B (Backup) salvare fişier existent înainte de copiere;
- O (Overwrite) fişierul existent este şters;
- S (Skip) se renunţă la această copiere;

A ştergere automată înainte de copiere
B fişierul existent va fi redenumit în <nume>. BAK
D copierea nu se face, dacă fişierul există deja pe discul destinaţie.
M se marchează fişierul original (>) şi copia (<)
Q copierea se face cu schimbarea numelui fişierului
T terminarea execuţiei dacă fişierul de copiat nu încape pe discul destinaţie (OFF: încearcă să copieze un fişier mai mic).

RUN[fişier parm]

Încarcă şi lansează în execuţie un program dintr-un fişier de tip COM cu transmitere opţională de parametri.

GO fişier adr parm

Încarcă un fişier la adresa <adr> şi îl lansează în execuţie, cu transmitere parametri prin <parm>.

LOAD fişier adr

Încarcă un fişier la adresa <adr>. După încărcarea fişierului se afişează limita superioară a memoriei ocupate.

EX adr arg

Lansează programul din memorie de la adresa <adr>, şi apoi se întoarce în POWER.

JP adr arg

Lansează programul din memorie de la adresa <adr>, iar după terminare program se întoarce în sistemul de operare. Prin <arg> se transmit parametrii programului.

SAVE fişier adr nsec

Salvare pe disc în fişierul ⟨fişier⟩ a zonei de memorie care începe la adresa ⟨adr⟩. Numărul de sectoare ⟨nsec⟩ necesare pentru fişier nu trebuie introdus dacă fişierul a fost încărcat în memorie cu POWER.

Comenzi de lucru în memorie

DUMPX {[adr], i} afişare memorie hexazecimal + ASCII;
DUMPH {[adr],,} afişare memorie în hexazecimal;
DUMPA {[adr],} afişare memorie în ASCII;

Dacă lipseşte adresa zonei de afişat ⟨adr⟩, atunci afişarea are loc de la adresa curentă.

,i vor fi afişaţi ⟨i⟩ octeţi;
,, vor fi afişaţi toţi octeţii următori.

Oprirea afişării se face cu tasta spaţiu, iar continuarea cu ⟨CR⟩. Întreruperea afişării se face cu CTRL/C.

FILL adr1 adr2 byte

Umple zona de la ⟨adr1⟩ la ⟨adr2⟩ cu octetul ⟨byte⟩.

MOVE adr1 adr2 adr3

Mută zona de memorie dintre ⟨adr1⟩ şi ⟨adr2⟩ la adresa ⟨adr3⟩.

DS adr

Afişarea conţinutului memoriei octet cu octet începînd cu adresa ⟨adr⟩, specificată în hexazecimal, zecimal, binar sau ASCII. Introducerea datelor se poate face în una dintre formele:

⟨H⟩ hexazecimal
⟨D⟩ zecimal
⟨B⟩ binar
⟨A⟩ ASCII

Forma de introducere a datelor se poate modifica prin comenzile: .H, .D, .B .A. Dacă nu se doreşte modificarea conţinutului afişat, atunci se apasă tasta ⟨CR⟩. Pentru schimbarea conţinutului memoriei se poate introduce o linie de pînă la 120 octeţi, terminată cu ⟨CR⟩ (octeţii trebuie separaţi prin caracterul spaţiu). Caracterele de control (OOH la 1FH) pot fi memorate cu secvenţa de caractere ^x, unde x reprezintă caracterul de control respectiv.

CM adr1 adr2 adr3

Compară octet cu octet conţinutul memoriei de la adresa ⟨adr 1⟩ pînă la adresa ⟨adr 2⟩ cu conţinutul memoriei de la adresa ⟨adr 3⟩. Diferenţele vor fi afişate cu adresă şi conţinut.

SEARCH adr1 adr2 byte

Caută o secvenţă de octeţi ⟨byte⟩ în zona dintre adresele ⟨adr1⟩ şi ⟨adr2⟩. Octeţii găsiţi vor fi afişaţi împreună cu adresa lor. Şirul ⟨byte⟩ se poate da astfel:

— în reprezentare şir de caractere: "DUM";

— în reprezentare hexazecimală, cu octeţii separaţi prin spaţiu: 54 53;

— caracterul '?' pentru orice octet din zonă. Numărul maxim de octeți în șir este 128.

Comenzi pentru citire/scriere fizică disc

READGR nr adr nsec

Citește în memorie la adresa ⟨adr⟩ un număr de ⟨nsec⟩ sectoare începînd de la grupul ⟨nr⟩ (aflat cu ajutorul comenzii GROUP). Dacă lipsește ⟨adr⟩ se consideră implicit adresa 80H; dacă lipsește ⟨nsec⟩ se consideră implicit un sector.

READGR nr tip nsec

Afișează pe ecran ⟨nsec⟩ sectoare începînd de la blocul ⟨nr⟩ în forma determinată de parametrul ⟨tip⟩:

XX: afișare hexazecimal și ASCII
XH: afișare hexazecimal
XA: afișare ASCII

WRITEGR nr adr nsec

Scrie pe disc ⟨nsec⟩ sectoare de la adresa ⟨adr⟩ începînd de la blocul disc ⟨nr⟩.

READ trk sec adr nsec

Citește ⟨nsec⟩ sectoare la adresa ⟨adr⟩ începînd din pista ⟨trk⟩ și sectorul ⟨sec⟩. Implicit se consideră nsec = 1 și adr = 80H.

READ trk sec tip nsec

Afișează ⟨nsec⟩ sectoare disc pe ecran în forma ⟨tip⟩:

XX: afișare hexazecimal și ASCII
XH : afișare hexazecimal
XA : afișare ASCII

WRITE trk sec adr nsec

Scrie pe disc ⟨nsec⟩ sectoare de la adresa ⟨adr⟩ în pista ⟨trk⟩, sectorul ⟨sec⟩. Implicit nsec = 1 și adr = 80H.

Comenzi pentru unități discuri

TEST

Testează sectoare disc (în afara pistelor sistem) și afișează:

— la fiecare sector bun un asterisc „*";
— numerele sectoarelor defecte.

Dacă în interiorul unui bloc se găsește un sector defect, pot fi utilizate următoarele funcții:

— SAVE Y/N toate sectoarele eronate vor fi ocupate de fișierul cu numele special „= = = = = = = = . = = =" = astfel încît blocurile defecte nu mai pot fi alocate;

— SHOW BAD FILES Y/N se afișează toate fișierele care conțin sectoare defecte;

— REPAIR Y/N încearcă rescrierea sectoarelor defecte pentru corectare;

— O sumă de control.

TESTS

TESTS spre deosebire de TEST include şi pistele de sistem.

STAT [d:]

Afişează spaţiul disc ocupat şi liber pentru toate unităţile instalate în POWER sau pentru unitatea de disc ⟨d⟩.

RESET d:

Reiniţializare unitate de disc ⟨d⟩.

DISK [d:]

Afişează parametrii-disc pentru unitatea implicită sau pentru unitatea ⟨d⟩.

4. EDITOARE DE TEXTE UTILIZATE ÎN CP/M

Editoarele de texte sînt programe utilitare specifice modului de lucru interactiv, care permit introducerea de programe sursă sau de texte diverse de la consolă în fișiere disc, precum și modificarea fișierelor text create.

Un fișier text este structurat pe linii de text de lungime variabilă (șiruri de caractere ASCII), separate prin perechea de caractere <CR> <LF>.

În CP/M există mai multe editoare de texte diferite între ele ca facilități oferite, ca mod de lucru și ca performanțe.

Primul editor de texte al sistemului CP/M a fost ED, foarte asemănător cu editorul EDIT din SFDX și cu alte editoare care pot opera cu console fără ecran (console cu imprimare pe hîrtie).

În prezent toate microcalculatoarele folosesc numai console cu ecran și astfel editoarele în mod ecran au înlocuit editoarele de tipul ED, datorită avantajelor oferite utilizatorilor.

Un editor mod ecran afișează în permanență pe ecran un fragment din fișierul editat, cuprinzînd de obicei peste 20 de linii de text, iar modificările în textul afișat se fac foarte comod și efectul lor este imediat vizibil, fără a folosi alte comenzi suplimentare de verificare, ca la editoarele de tip ED.

Editorul de texte Wordmaster (WM) a fost conceput numai pentru crearea și modificarea de fișiere care conțin programe sursă, dar asigură pentru acest scop toate facilitățile de editare necesare, fiind un program compact, ușor de folosit și cu un timp de răspuns foarte bun.

Editorul de texte Wordstar (WS) este mai mult decît un simplu editor de programe sursă, fiind un procesor de texte complex, destinat aplicațiilor de manipulare și prelucrare a documentelor sau materialelor scrise, destinate publicării. Wordstar include un număr mare de comenzi, dar dimensiunea apreciabilă a programului (segmentat) conduce la un timp de reacție sensibil mai mare decît în cazul editorului Wordmaster.

Caracterele de control folosite de utilizator diferă la cele două editoare mod ecran din CP/M, ceea ce face mai dificilă utilizarea lor în paralel; caracterele Wordstar se folosesc însă și în alte editoare sau programe care lucrează în mod ecran.

Mulți utilizatori preferă să folosească un singur editor de texte în toate împrejurările (pentru programe sau documentații) și atunci aleg programul Wordstar, care mai are în plus pentru începători și avantajul comenzilor afișate permanent pe ecran.

Toate editoarele de texte au o serie de caracteristici comune, dintre care menționăm:

— lungimea fișierelor editate nu este limitată de editor, ci numai de spațiul disponibil pe disc, deoarece editorul operează asupra unui segment din fișier adus într-o zonă text din memorie, citirea și scrierea în sau din zona text fiind realizată automat în WM și WS dar nu și la ED;

— editorul nu operează direct asupra fișierului de intrare, ci creează un fișier de lucru, cu același nume, dar cu extensia „.$$$", care, la terminarea normală a editării, primește numele fișierului editat; conținutul fișierului inițial este păstrat sub același nume (avînd extensia „.BAK"); în felul acesta întreruperea editării de texte nu afectează fișierul inițial;

— introducerea și modificarea de text se fac în zona text din memorie, astfel încît la întreruperea accidentală a editării este foarte probabil ca efectul multor comenzi de editare să nu fi operat și asupra fișierului de ieșire al editorului; soluția practică este salvarea periodică prin comenzi din zona text pe disc.

4.1. Editor de texte în mod comandă

4.1.1. Utilizarea editorului ED

Editorul de text ED se utilizează în cazul mașinilor CP/M care folosesc o consolă cu imprimare sau o consolă cu afișare, pentru care nu a fost instalat un editor de tip ecran.

Comanda ED poate avea două forme:

```
ED ufn
ED ufn d:
```

Numele de fișier „ufn", absolut necesar, desemnează fie un fișier nou, ce urmează a fi creat cu editorul de texte ED, fie un fișier deja existent, asupra căruia se vor face modificări și adăugări cu editorul de texte. Parametrul opțional „d:" arată că fișierul de ieșire se scrie pe unitatea de disc cu numele „d".

Orice comandă ED constă dintr-un cod mnemonic de o literă, eventual urmat de un șir de caractere și eventual precedat de un număr pozitiv sau negativ.

Este permisă scrierea mai multor comenzi într-o linie, fără separatori între ele, iar primul caracter ⟨CR⟩ produce executarea tuturor comenzilor din linia respectivă.

Unele comenzi sînt urmate de un şir de caractere (sau chiar de două şiruri pentru comanda S); ca terminator al şirului de caractere se utilizează fie caracterul CTRL/Z, fie terminatorul de comandă ⟨CR⟩(pentru comanda S primul şir se termină cu ^Z).

Majoritatea comenzilor pot fi precedate de un factor de repetiţie „n", care arată de cîte ori se execută comanda. Absenţa acestui număr implică o singură executare a comenzii. deci n = 1. Semnul care poate preceda unele comenzi indică sensul de acţiune a comenzii faţă de poziţia curentă a indicatorului: semnul „+" arată sensul înainte, semnul „—" arată sensul înapoi.

Editorul „ED" foloseşte o zonă de memorie care conţine o parte a fişierului editat (sau tot fişierul editat); mărimea acestei zone şi deci numărul liniilor de text aflate în memorie depinde de memoria disponibilă (de lungimea zonei TPA).

Editorul ED nu citeşte automat fişierul de intrare în memorie, şi este necesară comanda A (Append) sau N (Next) pentru aducerea de text din fişier în memorie.

Scrierea de text din memorie în fişierul de ieşire se face automat la terminarea editării prin comanda E (Exit) sau ca urmare a unor comenzi W (Write). Comanda W se foloseşte în cazul editării unor fişiere mari, care nu încap în întregime în zona text; în aceste cazuri se editează succesiv segmente de text citite prin comanda nA şi scrise apoi printr-o comandă nW.

Editorul ED foloseşte un indicator (un cursor) în cadrul zonei text din memorie, care poate fi deplasat prin comenzi de editare peste un număr de caractere sau de linii. Poziţa curentă a indicatorului poate fi vizualizată prin comanda T (Type), care afişează caracterele din linia curentă aflate la dreapta sau la stînga cursorului. Indicatorul se află mereu între două caractere din text.

Pe parcursul introducerii de comenzi ED sau de text se pot utiliza următoarele caractere de control:

CTRL/C abandonarea editării şi reiniţializare sistem;
CTRL/E sfîrşit fizic de linie (⟨CR⟩, ⟨LF⟩);
CTRL/I tabulare (salt la prima poziţe multiplu de 8);
CTRL/L sfîrşit logic de linie pentru comenzi lungi;
CTRL/U şterge linia curentă;
CTRL/Z terminator şir de caractere;
DEL şterge ultimul caracter introdus.

În cazul unor comenzi greşite sau imposibilităţii executării unor comenzi, editorul ED afişează unul dintre următoarele caractere cu valoare de mesaj de eroare:

? comandă greşită;

> zona text din memorie e plină sau un şir de caractere prea lung într-o comandă;

comanda nu poate fi executată de numărul de ori specificat (factor de repetiţie greşit).

4.1.2. Comenzile editorului ED

nA (Append lines = adăugare de linii)
Se citesc „n" linii din fişierul de intrare în zona text din memorie, după ultima linie existentă în memorie. Comanda 0A citeşte tot fişierul de intrare sau pînă la umplerea zonei text.

B/—B (Begin/Bottom of buffer = început/sfîrşit de text)
Comanda B poziţionează indicatorul la începutul zonei text (înaintea primei linii), iar comanda —B poziţionează indicatorul la sfîrşitul zonei text (după ultima linie).

nC / —nC (Character forward/backward = caracter înainte/înapoi)
Se mută indicatorul peste „n" caractere înainte sau înapoi (—) faţă de poziţia curentă.

nD / — nD (Delete characters = şterge caractere)
Se şterg „n" caractere înainte (+) sau înapoi (—) faţă de poziţia curentă a indicatorului

E (Exit = ieşire)
Se închid fişierele de intrare şi de ieşire şi se termină editarea cu revenire în sistem.

nFtext ↑Z (Find string = caută şir)
Caută în zona text din memorie şirul de caractere conţinut în comandă (terminat cu <CR> sau cu CTRL/Z) şi se poziţionează indicatorul după textul găsit.

H
Terminare editare, închidere urmată de redeschiderea fişierelor.

Itext ↑Z (Insert = introducere de caractere)
Se introduce şirul de caractere conţinut în comandă începînd din poziţia curentă a indicatorului (înaintea caracterului curent). Comanda I<CR> trece editorul în mod „inserare", mod în care se pot introduce mai multe linii de text terminate cu <CR>; ieşirea din modul „inserare" se face prin caracterul CTRL/Z.

nK (Kill lines = şterge linii)
Se şterg „n" linii înainte (+) sau înapoi (—) faţă de poziţia curentă a indicatorului.

nL/—nL (Lines down/up = avans linii înainte/înapoi)
Se mută indicatorul peste „n" linii înainte (+) sau înapoi (—).

nNtext ↑Z (Next string = căutare de şir)
Caută textul conţinut în comandă în tot fişierul de intrare (dacă este necesar se citesc şi se scriu automat linii de text).

O (Original file = fișier primar)
Se revine la fișierul de intrare inițial.

nP/—nP (Print pages = imprimare pagini)
Afișează „n" pagini înainte (+) sau înapoi (—) față de poziția curentă.

Q (Quit edit = abandon editare)
Abandonează editarea fără a modifica fișierul de intrare.

nStext1^Ztext2^Z (Substitute string = înlocuire de caractere)
Se caută în zona text șirul de caractere ⟨text1⟩ și se înlocuiește cu șirul ⟨text2⟩ (echivalent cu secvența F, D, I).

nT/—nT (Type = afișare linii)
Afișează „n" linii înainte (+) sau n linii înapoi (—) față de poziția curentă a indicatorului. Comanda T afișează caracterele din linia curentă aflate la dreapta cursorului, comanda 0T afișează caracterele din linia curentă aflate la stînga cursorului, iar secvența 0TT afișează toată liniar curentă, indiferent de poziția cursorului în linie.

U/—U (Upper/lower case = litere mari/mici)
Comanda U are ca efect transformarea automată a literelor mici în litere mari, iar comanda —U oprește trecerea literelor mici în litere mari

nW (Write lines = scrie linii)
Scrie următoarele „n" linii din zona text în fișierul d ieșire.

n/—n
Mută indicatorul și afișează „n" linii înainte (+) sau înapoi (—) (echivalent cu nLT/—nLT).

4.2. Editorul de texte în mod ecran Wordmaster

4.2.1. Utilizarea editorului Wordmaster în mod ecran

Editorul WM are două moduri de lucru:
— modul ecran sau video;
— modul comandă.

În modul comandă, WM acceptă aproape toate comenzile editorului de texte ED, precum și alte comenzi suplimentare, putînd fi considerat ca o dezvoltare a programului ED.

În modul ecran, comenzile WM au forma unor caractere de control de tipul CTRL/x (↑x).

Fișierul WM.COM ce conține editorul Wordmaster este însoțit de obicei de un fișier WM.HLP, care conține toate comenzile WM într-o formă condensată. Acest breviar de comenzi poate fi afișat la consolă și din cadrul editorului WM prin comanda ↑Q.

Apelarea editorului de text WM se face prin comanda:

WM d: nume.tip

unde „d: nume. tip" este numele unui fişier existent sau care urmează a fi creat (numele de fişier nu poate lipsi din comandă).

Imediat după lansare WM trece în modul video şi, dacă se editează un fişier existent, fişierul de intrare este citit automat în memorie, iar prima pagină din fişier este afişată pe ecran.

O pagină de text conţine cel mult 23 de linii, ultima linie de pe ecran fiind rezervată pentru introducerea de comenzi în modul comandă.

În modul video există comenzi pentru deplasarea cursorului în cadrul paginii afişate pe ecran, comenzi pentru modificarea textului afişat şi comenzi pentru schimbarea paginii curente.

În mod video pot exista două submoduri:

— *mod înlocuire*, în care un caracter tastat înlocuieşte caracterul din dreptul cursorului;

— *mod inserare*, în care un caracter tastat este inserat în poziţia curentă a cursorului, după deplasarea la dreapta a caracterelor aflate între cursor şi sfîrşitul liniei.

Iniţial editorul WM este în modul înlocuire; comutarea pe inserare şi înapoi se face cu caracterul CTRL/F. Modul inserare este semnalat prin afişarea caracterului '>' în dreptul cursorului, făcînd invizibil caracterul curent.

Trecerea din mod ecran în mod comandă se face prin tastarea caracterului ESC („Escape"), iar trecerea din mod comandă în mod video se face prin comanda V (Video Mode).

Ieşirea din WM se poate face numai în modul comandă, prin comanda E (Exit) sau Q (Quit).

Comenzi WM pentru modul video. Comenzi pentru *deplasarea cursorului:*

CR mută cursorul la începutul liniei următoare;
LF mută cursorul în jos cu o linie;
CTRL/H deplasare stînga un caracter (BS);
CTRL/L deplasare dreapta un caracter;
CTRL/K deplasare în sus cu o linie;
CTRL/J deplasare în jos cu o linie (LF);
CTRL/M deplasare în jos pe început de linie (CR);
CTRL/Z deplasare dreapta peste TAB;
CTRL/A deplasare stînga peste un cuvînt;
CTRL/D deplasare dreapta peste un cuvînt;
CTRL/B deplasare la început sau sfîrşit de linie;
CTRL/T deplasare la început sau sfîrşit de pagină.

Comenzi pentru *modificarea de text:*

DEL şterge caracter la stînga;
CTRL/G şterge caracter la dreapta;

CTRL/\ şterge cuvînt la stînga;
CTRL/O şterge cuvînt la dreapta;
CTRL/U şterge linie la stînga;
CTRL/P şterge linie la dreapta;
CTRL/Y şterge linie în întregime;
CTRL/I inserează un TAB în text (TAB);
CTRL/N inserează un terminator de linie (CR, LF);
CTRL/F activare (dezactivare) mod inserare.

Comenzi pentru *schimbarea paginii* afişate pe ecran:
CTRL/C pagina următoare;
CTRL/R pagina precedentă;
CTRL/X deplasare în sus cu o linie;
CTRL/E deplasare în jos cu o linie.

Comenzi *diverse:*
CTRL/Q afişare listă de comenzi;
ESC ieşire din mod ecran.

4.2.2. Utilizarea Wordmaster în mod comandă

Modul comandă nu se foloseşte pentru editarea propriu-zisă a textului din memorie, ci pentru operaţii mai speciale, cum ar fi: căutarea automată a unui text în zona text sau în tot fişierul de intrare, căutare şi înlocuire de text, citirea sau scrierea altor fişiere, mutarea unor blocuri de text în altă poziţie, ieşirea din editor ş.a.

În general comenzile WM sînt aceleaşi cu comenzile ED, dar există şi unele diferenţe prezentate în continuare:

— ca terminator de text într-o comandă se poate folosi şi caracterul ESC pe lîngă caracterele CTRL/Z şi CR;

— caracterul ⟨CR⟩ singur nu poate fi folosit pentru trecere la linia următoare, dar poate fi folosit sub forma 1⟨CR⟩ sau —⟨CR⟩ sau n ⟨CR⟩;

— comanda A (Append) este similară comenzii I (Insert), dar textul nou adăugat se inserează după linia curentă, indiferent de poziţia indicatorului în linia curentă;

— citirea dintr-un fişier de intrare se face prin comanda Y (Yank), dar nu linie cu linie, ci tot fişierul în întregime; se pot citi mai multe fişiere în cursul unei editări şi deci se pot concatena sau combina fişiere cu WM;

— înlocuirea de text se poate face şi prin comanda R (Replace), care, spre deosebire de comanda S (Substitute), caută şirul ce trebuie înlocuit în tot fişierul de intrare dacă nu este găsit în zona text din memorie;

— comanda W (Write) conţine şi numele fişierului în care se scrie, deci se pot crea mai multe fişiere de ieşire;

— comenzile de căutare-înlocuire (F, N, S, R) pot fi precedate de caracterul „/" care produce poziţionarea pe sfîrşitul textului (fişie-

rului) în care se caută; în mod normal, la o căutare nereușită nu se modifică poziția indicatorului și se afișează „ ++ ";

— editorul WM folosește o zonă de manevră („Scratchpad") în care se pot introduce linii de text sau comenzi de editare. Această zonă se poate utiliza fie pentru transferul unor linii de text dintr-o parte în alta a fișierului editat, fie pentru crearea și executarea de macrocomenzi de editare. Exemplu de definire a unei macrocomenzi pentru afișarea liniei curente în întregime, indiferent de poziția indicatorului:

QL0TT

Caractere de control utilizabile în comenzile de căutare:

CTRL/A orice caracter;
CTRL/AX orice caracter egal cu caracterul X;
CTRL/OX orice caracter diferit de caracterul X;
CTRL/N CR, LF;
CTRL/Y ESC.

Lista comenzilor editorului WM (mod comandă)

A sau nAtext (Append)
Intră în mod inserare sau adaugă textul specificat după linia curentă.

nC/—nC (Character)
Mută indicatorul peste „n" caractere înainte/înapoi.

nD/—nD (Delete)
Sterge „n" caractere la dreapta/stînga indicatorului.

E (Exit)
Terminare editare.

nFtext/—nFtext (Find)
Caută a „n"-a apariție a lui „text" în zona text de memorie.

H
Termină editarea și reia editarea de la început.

I (Insert)
Intră în modul inserare; se iese cu ^Z sau <ESC>.

nK/—nK (Kill)
Sterge „n" linii înainte (înapoi).

nL / —nL (Lines)
Mută indicatorul peste „n" linii înainte (înapoi).

nNtext / —nNtext (Next)
Caută a „n"-a apariție a lui „text" în fișierul de intrare.

O (Original file)
Revenire la fișierul inițial.

nP / —nP (Page)
Mută indicatorul peste „n" pagini și afișează pagina curentă.

Q (Quit)
Abandonare editare fără modificare fișier de intrare.

nQP (Put)
Transferă următoarele „n" linii în zona de manevră și șterge aceste linii din fișier.

n/QP
Adaugă următoarele „n" linii la zona de manevră şi le şterge din fişier.

nQG (Get)
Transferă de „n" ori conţinutul zonei de manevră în fişier.

QT (Type)
Afişează conţinutul zonei de manevră.

QK (Kill)
Şterge conţinutul zonei de manevră.

QX (Execute)
Execută comenzile din zona de manevră.

QLtext (Load)
Introduce „text" în zona de manevră.

n/QLtext
Adaugă de „n" ori „text" la zona de manevră.

+/—nRtext1 ^Ztext2 (Replace)
Înlocuieşte de „n" ori „text1" prin „text2" în tot fişierul.

+/—nStext1 ^Ztext2 (Substitute)
Înlocuieşte de „n" ori înainte (înapoi) „text1" prin „text2" în zona de text din memorie.

nT / —nT (Type)
Afişează „n" linii de text înainte (înapoi) (din linia curentă se afişează numai caracterele dintre cursor şi sfîrşitul liniei).

V (Video)
Intrare în mod video (mod ecran).

Yd: nume. tip (Yank)
Citeşte fişierul specificat şi introduce conţinutul acestuia începînd cu poziţia curentă a cursorului.

nWd: nume. tip (Write)
Scrie „n" linii în fişierul specificat.

Observaţii:

— este posibil ca factorul de repetiţie „n" să fie dat acoperitor mult mai mare decît numărul de linii din fişier deoarece editorul se opreşte la sfîrşitul fişierului sau zonei text; această observaţie poate fi utilă la comenzile „A", „L", „S", „R", „W" ş.a.;

— pentru mutarea unui bloc de text dintr-un loc în altul se va utiliza zona de manevră astfel:

se poziţionează cursorul pe începutul blocului;
se scriu liniile de text în zona de manevră (nQP);
se poziţionează cursorul pe locul unde se mută;
se citeşte blocul de text din zona de manevră (QG);

— dacă la comanda „E" de ieşire din WM apare mesajul de disc plin („Disk Full"), atunci se poate folosi comanda „W" pentru a scrie textul din memorie peste un alt fişier de pe acelaşi disc pentru a nu se pierde rezultatul editării. Fişierul distrus trebuie să fie suficient de mare şi nu trebuie să fie protejat la scriere.

4.3. Editorul de texte mod ecran Wordstar

4.3.1. Prezentare generală a produsului Wordstar

Există cîteva versiuni succesive ale programului Wordstar, aici fiind comentată versiunea 3.0, care necesită fişierele :

WS3.COM	rădăcina Wordstar (sau WS.COM);
WSOVLY3.OVR	segment cu instrucţiuni;
WSMSGS.OVR	segment cu mesaje.

Numele sub care apare fişierul rădăcină poate fi diferit, adăugîndu-i-se uneori numele maşinii sau terminalului pentru care a fost instalat (instalarea nu afectează şi segmentele OVR).

Din Wordstar se pot apela explicit, de către utilizator, alte două segmente opţionale pentru operaţii mai speciale de imprimare cu interclasare (MAILMRGE.OVR) sau pentru verificare sintactică de texte în limba engleză (SPELLSTAR).

Apelarea programului Wordstar se poate face în cîteva moduri, dar care nu sînt echivalente:

WS

pentru încărcare în memorie.

WS fişier

pentru încărcare şi intrare în mod editare document ; fişierul de ieşire se creează pe discul unde se află fişierul de intrare.

WS fişier d:

pentru editare document cu crearea fişierului de ieşire pe un alt disc decît fişierul de intrare; ⟨fişier⟩ reprezintă specificatorul fişierului de intrare, din care numele discului poate lipsi dacă este discul implicit.

Programul Wordstar se poate afla în mai multe stări, iar fiecare stare este caracterizată printr-o anumită listă de *comenzi* afişat pe ecran. În fiecare stare se pot folosi numai caracterele de comandă afişate în meniul stării respective, cu semnificaţia prezentată. Este posibil ca un acelaşi caracter să aibă interpretări diferite în meniuri diferite sau ca o aceeaşi operaţie să se poată cere prin comenzi diferite sau prin aceeaşi comandă în meniuri diferite.

Caracterele care nu sînt permise într-o stare sînt ignorate neavînd nici un efect.

Trecerea dintr-o stare în alta se face de regulă prin tastarea unui caracter de control de către utilizator; după executarea unei comenzi se revine automat în starea din care s-a emis comanda.

O funcţie Wordstar poate fi solicitată fie printr-un singur caracter, dacă ne aflăm în starea care acceptă funcţia respectivă, fie printr-o secvenţă de două caractere dacă dorim o funcţie existentă în altă stare decît starea curentă. De exemplu, din lista principală de comenzi putem deplasa cursorul cu o poziţie folosind un singur caracter, dar pentru de-

plasare la sfîrşit de fişier trebuie folosite două caractere: unul pentru a trece în altă listă de comenzi (↑ Q) şi altul pentru comanda de mutare (C sau ^C). De observat că cel de al doilea caracter (numele comenzii) poate fi introdus ca literă sau ca un caracter de control derivat din aceeaşi literă.

Programul Wordstar este gîndit să se adapteze nivelului de experienţă al utilizatorului în două feluri:

— prin nivelul de ajutor, sub formă de meniuri afişate pe ecran;

— prin temporizările introduse în realizarea unor operaţii.

De exemplu, dacă nu ştim cum se poate ajunge la sfîrşit de fişier atunci activăm nivelul maxim de ajutor (nivelul 3), tastăm ↑ Q şi aşteptăm afişarea listei de comenzi din noua stare (Quick Menu) ca să aflăm numele comenzii de poziţionare pe sfîrşit (C); dacă ştim însă comanda, atunci putem introduce cele două caractere (↑ QC) fără pauză între ele, eliminînd timpul de afişare a listei de comenzi ↑ Q.

Nivelul de ajutor (Help) poate fi 0,1,2 sau 3; la nivelul 0 nu se afişează numele comenzilor utilizabile iar la nivelul 3 se afişează permanent pe ecran numele comenzilor din fiecare meniu, prin reducerea paginii de text afişate pe ecran cu 8 linii.

Indiferent de nivelul de ajutor în editare se afişează pe ecran o linie de stare şi o linie de reper (Ruler line). Linia de stare conţine numele comenzii, numele fişierului editat, numărul paginii, numărul liniei, coloanei şi modul inserare (înlocuire). Linia de reper indică limita din stînga şi limita din dreapta (L,R), precum şi poziţiile de tabulare implicite (unde se sare la apăsarea pe clapa TAB). Dacă editarea a fost iniţiată cu comanda N, atunci numerele de pagină, linie şi coloana sînt înlocuite cu:

FC — numărul caracterului în fişier (inclusiv ⟨CR⟩);

FL — numărul liniei în fişier.

Stările şi meniurile de comenzi Wordstar sînt următoarele:

Lista de comenzi „fără fişier" (No-file Menu) conţine următoarele categorii de comenzi:

— comenzi pentru deschidere de fişiere în vederea editării;

— comenzi de copiere, ştergere, redenumire şi tipărire fişier;

— comenzi pentru execuţia altor programe sau opţiuni Wordstar;

— comenzi la nivel de disc: schimbare disc implicit, afişare director;

— stabilire nivel de ajutor;

— ieşire din Wordstar.

Lista de comenzi principală (Main Menu) conţine următoarele categorii de comenzi:

— comenzi pentru deplasare cursor şi defilare ecran;

— comenzi de ştergere caractere şi linii;

— comenzi diverse: comutare între inserare şi înlocuire, reformatare paragraf, inserare caractere speciale;

— comenzi pentru comutare în alte liste de comenzi;

Lista de comenzi rapidă (Quick Menu) conține următoarele categorii de comenzi:

— comenzi pentru deplasare cursor pe distanțe mai mari: început sau sfîrșit de linie, de pagină, de bloc, de fișier, pe marcaje;

— comenzi de ștergere parțială linie, la stînga sau la dreapta;

— comenzi de căutare și de înlocuire;

— comenzi pentru comutare în alte liste de comenzi

Lista de comenzi pe blocuri (Block Menu) conține următoarele categorii de comenzi:

— comenzi de salvare text editat și terminare editare;

— comenzi de plasare marcaje în text;

— comenzi de marcare început și sfîrșit de bloc;

— comenzi pe blocuri: copiere, mutare, ștergere, scriere în fișier

— comenzi pentru fișiere: citire, copiere, ștergere, redenumire, imprimare;

— comenzi pentru discuri: schimbare disc implicit, afișare tabel director.

Lista de comenzi imprimare (Print Menu) conține comenzi care se introduc în text pentru obținerea unor efecte speciale la imprimarea acestui text: litere îngroșate prin dublare, subliniere automată, indici, exponenți, supraimprimare, alegerea formei de caractere și a densității de scriere, alegerea culorii etc.

Lista de comenzi ecran (Onscreen Menu) conține comenzi pentru:

— modificarea marginilor din stînga și din dreapta a liniilor sau pentru desființarea marginilor;

— stabilirea pozițiilor de tabulare (de salt);

— centrare text în linie;

— stabilire interval între linii;

— controlul mutării automate a cuvintelor pe linia următoare;

— controlul despărțirii cuvintelor în silabe;

— controlul paginării textului.

Lista de comenzi de ajutor (Help Menu) conține comenzi pentru afișarea unor informații detaliate despre:

— stabilirea nivelului de ajutor;

— reformatare paragrafe;

— caractere de control în ultima coloană;

— comenzi cu punct, pentru controlul imprimării;

— linia de stare și linia de reper;

— margini și poziții de tabulare;

— marcaje;

— deplasare text.

Starea inițială a programului Wordstar depinde de modul în care a fost apelat programul:

— apelarea fără fișier parametru trece editorul în meniul „fără fișier";

— apelarea cu un parametru fișier trece editorul în lista principală de comenzi și în starea editare document, echivalent cu comanda D din lista de comenzi „fără fișier".

Apelarea fără fișier se folosește de exemplu pentru editarea în mod nedocument (de programe sursă) și pentru imprimarea de fișiere editate anterior.

4.3.2. Utilizare Wordstar ca editor de programe

În acest subcapitol sînt prezentate numai comenzile necesare editării de programe sursă, comenzi grupate în următoarele trei liste de comenzi:

— lista de comenzi principală introdusă prin comanda N;
— lista de comenzi rapide, introdusă prin ↑ Q;
— lista de comenzi bloc, introdusă prin ↑ K.

Introducerea și editarea de programe sursă se poate face și în modul editare document (comanda D), dar este preferabil să se facă în modul editare nedocument (comanda N) din următoarele motive:

— fixarea limitelor de tabulare (din 8 în 8), precum și alte opțiuni de lucru Wordstar au valori implicite similare cu alte editoare de texte (nu se împarte automat textul în pagini, nu se mută automat cuvinte de pe o linie pe alta);

— fișierele create în modul nedocument sînt complet compatibile cu alte programe CP/M, inclusiv cu celelalte editoare de texte ED și WM și cu utilitarul PIP, spre deosebire de fișierele create cu comanda D, care au anumite particularități (de exemplu, bitul superior din anumite caractere poate fi diferit de zero).

În modul document pozițiile de tabulare dintr-o linie sînt așezate din 5 în 5 caractere.

Urmează o scurtă descriere a comenzilor din cele trei liste de comenzi menționate.

Lista de comenzi „fără fișier":

L (Change Logged disk drive)
Modificare unitate de disc implicită pentru fișierele editate.

F (File directory on/off)
Activează sau dezactivează afișarea pe ecran a tabelului director pentru discul implicit.

H (Set Help Level)
Stabilirea nivelului de ajutor la 0, 1, 2 sau 3. Implicit este 3.

D (Open a Document file)
Deschide fişier document pentru creare sau modificare.
N (Open a Non-document file)
Deschide fişier nedocument pentru creare sau modificare.
P (Print a file)
Imprimă un fişier, iar după începerea imprimării are efectul de oprire a tipăririi la imprimantă. Caracterele TAB sînt expandate prin spaţii pînă la următoarea limită de 8 spaţii.
E (rEname a file)
Schimbă numele unui fişier.
O (cOpy a file)
Copiază un singur fişier. Dacă există deja un fişier cu numele fişierului destinaţie, atunci se cere confirmare. ⟨CR⟩ sau CTRL/U anulează comanda de copiere.
R (Run a program)
Cere executarea unui program, după care se revine în Wordstar.
X (eXit)
Ieşire din Wordstar în CP/M.
Lista principală de comenzi:
↑S (char left)
Mută cursor la stînga cu un caracter.
↑D (char right)
Mută cursor la dreapta cu un caracter.
↑A (word left)
Mută cursor la stînga cu un cuvînt.
↑F (word right)
Mută cursor la dreapta cu un cuvînt.
↑E (line up)
Mută cursor în sus cu o linie, pe aceeaşi poziţie de caracter.
↑X (line down)
Mută cursor în jos cu o linie, pe aceeaşi poziţie de caracter.
↑Z (scroll line up)
Defilare text în sus u o linie.
↑W (scroll line down)
Defilare text în jos cu o linie.
↑C (scroll screen up)
Defilare text în sus cu o pagină ecran (pagina următoare).
↑R (scroll screen down)
Defilare text în jos cu o pagină ecran (pagina precedentă).
↑G (delete char)
Şterge caracterul din dreptul cursorului.
DEL (del char left)
Şterge caracter la stînga cursorului.
↑T (del word right)
Şterge cuvînt la dreapta cursorului.

↑Y (delete line)
Şterge linia pe care se află cursorul.

↑B (reform)
Reformatare paragraf.

↑V (insert on/off)
Comută între modul inserare şi modul înlocuire (implicit inserare).

↑L (find/replace again)
Repetă ultima comandă de căutare (înlocuire) (↑QF sau ↑QA).

Lista de comenzi de blocuri:

↑KS (save & resume)
Salvare text din memorie pe disc şi reluare editare de la început fişier (se poate reveni în poziţia anterioară salvării cu ↑QP).

↑KD (save, done)
Salvare text din memorie pe disc şi terminare editare fişier curent, fără ieşire din Wordstar.

↑KX (save & exit)
Salvare text din memorie pe disc, terminare editare şi ieşire în CP/M.

↑KQ (abandon file)
Abandonare editare şi restaurare fişier iniţial.

↑K0..↑K9 (set/hide markers)
Introducere sau ştergere marcaje cu numere între 0 şi 9.

↑KB (begin of block).
Marcare început de bloc.

↑KK (end of block)
Marcare sfîrşit de bloc şi punere în evidenţă bloc pe ecran.

↑KH (hide/display)
Anulează marcajele de bloc şi evidenţierea blocului pe ecran.

↑KC (copy block)
Copiază bloc în poziţia cursorului; blocul rămîne şi în poziţia sa iniţială.

↑KV (moVe block)
Mută bloc în poziţia curentă a cursorului, cu ştergere din poziţia sa iniţială.

↑KY (delete block)
Şterge blocul marcat din fişier.

↑KW (write block)
Scrie blocul marcat într-un fişier pe disc. Nu se pot concatena mai multe blocuri într-un singur fişier.

↑KR (read file)
Citeşte text dintr-un fişier în poziţia cursorului.

↑KO (copy file)
Copiază un fişier.

↑KJ (delete file)
Şterge un fişier.

↑KE (rename file)
Schimbă numele unui fişier.
↑KP (print file)
Scrie la imprimantă conţinutul unui fişier.
↑KL (change logged disk)
Modifică discul implicit.
↑KF (directory on/off)
Activează sau dezactivează afişarea tabelei directoare.
Lista de comenzi de operaţii rapide:
↑QS (left side)
Mută cursor la începutul liniei curente (în marginea stîngă).
↑QD (right side)
Mută cursor la sfîrşitul liniei curente (în marginea dreaptă).
↑QE (top screen)
Mută cursor pe început de ecran (pe prima linie).
↑QX (bottom screen)
Mută cursor pe sfîrşit de ecran (pe ultima linie).
↑QR (top file)
Mută cursor pe început de fişier editat.
↑QC (end file)
Mută cursor pe sfîrşit de fişier editat.
↑QB (top block)
Mută cursor pe început de bloc.
↑QK (end block)
Mută cursor pe sfîrşit de bloc.
↑Q0..↑Q9 (marker 0—9)
Mută cursor pe unul dintre marcajele 0...9.
↑QP (previous)
Mută cursor pe poziţia precedentă, unde se afla înainte de ultima comandă.
↑QZ (scroll up)
Defilare continuă în sus a textului pe ecran.
↑QW (scroll down)
Defilare continuă în jos a textului pe ecran.
↑QV (last find or block)
Mută cursor pe poziţia unde se afla înaintea unei comenzi de căutare sau de înlocuire.
↑QY (line right)
Şterge linie la dreapta cursorului.
↑QDEL (line left)
Şterge linie la stînga cursorului.
↑QF (find text in file)
Caută un şir dat în fişierul de intrare începînd din poziţia cursorului şi poziţionează cursor pe sfîrşit de şir. Următoarele apariţii ale aceluiaşi şir pot fi localizate prin ↑L.

↑ QA (find & replace)
Caută și înlocuiește un șir dat cu alt șir dat în fișierul de intrare, începînd din poziția cursorului. Continuarea comenzii se poate face cu ↑L.

↑ QQ (repeat command)
Repetă comanda următoare, pînă cînd se apasă pe o tastă de la consolă.

Opțiuni la căutare și înlocuire:

n (unde n este un număr cuprins între 1 și 65 535) caută a n-a apariție a șirului căutat sau execută operația de înlocuire de n ori;

G execută operația de înlocuire pentru toate aparițiile șirului de înlocuit în fișier, sau caută ultima apariție a șirului în fișier;

N înlocuiește șirul căutat fără acordul utilizatorului;

B execută operația de căutare și (sau) înlocuire de la poziția cursorului spre începutul fișierului;

U ignoră diferența dintre literele mari și cele mici în șirul de căutat;

W caută sau înlocuiește numai cuvinte întregi.

Observații

— Inițial editorul Wordstar este în modul inserare, ceea ce permite introducerea de text nou sau inserarea de caractere între alte caractere existente în text. Pentru înlocuirea de caractere se poate trece temporar în modul „înlocuire" prin ↑V.

— Înlocuirea repetată a unor caractere sau a unor șiruri de caractere se poate face prin comanda ↑QA, introducînd ca opțiuni numărul de înlocuiri dorite (eventual un număr acoperitor, deoarece implicit se cere confirmarea operatorului la fiecare înlocuire).

— De reținut că o căutare sau o înlocuire se face în tot fișierul, fie după poziția cursorului (spre sfîrșitul fișierului), fie înainte de poziția cursorului (spre începutul fișierului), dacă se folosește opțiunea B.

— Un bloc de text este un șir de caractere de orice lungime (de la un singur caracter pînă la mai multe linii de text), delimitat prin marcaje de bloc stabilite de utilizator. Marcajele de început și de sfîrșit bloc pot fi modificate și pot fi anulate (prin ↑KH). Asupra unui bloc de text marcat se pot face diverse operații: copiere, mutare, ștergere, scriere într-un alt fișier.

— Mutarea unui bloc de text pe o distanță mai mică se face astfel:

— se aduce cursorul pe începutul blocului și se tastează ↑KB;

— se mută cursorul pe sfîrșitul blocului și se tastează ↑KK;

— se mută cursorul în poziția unde trebuie adus blocul și se tastează ↑KV sau ↑KC, după cum se dorește mutarea sau copierea blocului marcat;

— se tastează ↑KH pentru anularea marcajelor de bloc, dacă nu se fac mai multe copieri ale unui bloc.

— Mutarea unui bloc de text pe o distanță mai mare se face mai rapid astfel:

— se stabilesc marcajele de început și de sfîrșit de bloc;
— se scrie blocul marcat într-un fișier disc (prin ↑KW);
— se mută cursorul acolo unde trebuie mutat blocul;
— se citește din fișierul creat anterior (prin ↑KR);
— se șterge fișierul de manevră (prin ↑KJ).

5. PROGRAME PENTRU UTILIZAREA LIMBAJULUI MAŞINĂ

Limbajul numărul unu pentru microcalculatoare pe 8 biţi este limbajul de asamblare, adică limbajul constituit din instrucţiunile maşină, la care se adaugă un număr de directive de asamblare.

De fapt, sub CP/M există două limbaje de asamblare, după cum se utilizează codurile simbolice INTEL, pentru procesorul 8080, sau codurile mnemonice ZILOG, pentru procesorul Z80.

Deoarece codul intern al instrucţiunilor comune este acelaşi pentru cele două microprocesoare, de multe ori programele pentru Z80 se scriu cu mnemonice de 8080, eventual folosind şi instrucţiunile suplimentare Z80, generate ca date constante cu directive.

În prezent există sub sistemul CP/M mai multe programe asamblor, care acceptă cam acelaşi limbaj de asamblare (cu mnemonice 8080), dar care se utilizează în moduri diferite şi generează fişiere obiect de formate diferite.

Primul asamblor pentru CP/M a fost ASM (Digital Research), un asamblor care generează cod obiect cu adrese absolute în format hexazecimal Intel.

Programul MAC (Digital Research) este un macroasamblor rezultat prin adăugarea la ASM a posibilităţii de a defini şi utiliza macroinstrucţiuni.

Programul M80 (Microsoft) este un macroasamblor care generează cod obiect relocabil (cu adrese modificabile) şi este însoţit de către editorul de legături L80. Acelaşi format relocabil generat de Macro-80 rezultă şi după compilare Fortran sau după alte compilatoare Microsoft (BASIC, COBOL).

Formatul relocabil Microsoft a devenit principalul format pentru fişierele obiect sub sistemul CP/M, deşi ulterior au mai fost introduse şi alte formate obiect relocabile, asociate unor limbaje de nivel înalt.

Programul RMAC (Digital Research) este un macroasamblor care generează tot cod relocabil Microsoft, deci foloseşte tot linkeditorul Link-80, dar se utilizează ca programele MAC şi ASM.

Asamblorul AS (Manx) a fost scris pentru prelucrarea codului generat de compilatorul CII, dar poate fi folosit și separat, împreună cu linkeditorul LN, care oferă anumite facilități pentru segmentarea programelor.

Pentru crearea, modificarea și inventarierea bibliotecilor de module obiect în format relocabil se folosesc programe utilitare de tip bibliotecar, care diferă după formatul relocabil prelucrat.

Depanarea programelor scrise în limbaj mașină se face sub controlul unui program care este folosit în mod interactiv pentru vizualizarea registrelor și memoriei, pentru modificarea datelor sau programului și pentru execuția pe porțiuni sau instrucțiune cu instrucțiune a programului depanat.

Primul program depanator sub CP/M a fost DDT (Digital Research), din care au rezultat apoi două depanatoare simbolice: SID pentru 8080 și ZSID pentru Z80.

5.1. Programe asamblor cu generare de cod obiect absolut

Un program asamblor este un translator de programe din format sursă (limbaj de asamblare) în format obiect (cod intern mașină).

Utilizarea unui asamblor absolut sub CP/M poate fi necesară în următoarele situații:

— trebuie obținut un fișier obiect în format hexazecimal, care să se încarce la o adresă diferită de 100H (de exemplu, la asamblarea componentei BIOS din nucleul CP/M);

— se dorește depanarea simbolică a programului asamblat cu SID și deci este necesar un fișier tabelă de simboluri;

— timpul de punere la punct a unui program trebuie să fie cît mai mic și nu este necesară o fază de linkeditare după asamblare.

Asamblorul absolut are avantajul unui timp minim de obținere a unui program executabil (HEX sau COM) și al producerii de adrese absolute direct utilizabile în depanarea programelor.

Principalul dezavantaj al folosirii unui asamblor absolut este acela că după orice corectură efectuată în program, este necesară reasamblarea întregului program, care se păstrează într-un singur fișier sursă; acest neajuns este resimțit în cazul programelor mai mari, care depășesc cîteva sute de linii sursă.

5.1.1. Cod obiect absolut

Sarcina principală a unui asamblor este să treacă instrucțiunile mașină din forma simbolică, externă, în forma binară, internă.

Pentru instrucțiunile cu referire directă la memorie, asamblorul trebuie să înlocuiască o adresă simbolică (care reprezintă eticheta unei

alte instrucţiuni sau a unei directive) cu o adresă numerică, care face parte din codul intern al instrucţiunii.

În cazul cel mai simplu, adresele generate de asamblor sînt adrese de memorie absolute, definitive. Prin cod obiect absolut se înţelege un cod obiect generat de un asamblor, care conţine adrese de memorie absolute. Nu este însă obligatoriu ca acest cod să coincidă cu formatul programelor direct executabile, numit şi format imagine memorie.

În sistemul CP/M cele două asambloare care generează cod obiect absolut (ASM şi MAC) produc fişiere în format hexazecimal şi nu în format binar.

Programele din fişierele HEX pot fi încărcate direct şi executate cu ajutorul instrumentelor de depanare DDT, SID, ZSID; această operaţie se practică în cursul testării şi punerii la punct a programelor scrise în limbaj de asamblare.

Programele verificate şi care urmează a fi executate direct prin comenzi consolă trebuie trecute din format hexazecimal în format direct executabil (COM), cu ajutorul programului LOAD.

Programul LOAD se apelează prin comanda

LOAD nume sau LOAD d: nume

unde ,,nume'' este numele fişierului hexazecimal, tipul fişierului fiind implicit HEX.

Rezultatul comenzii LOAD este crearea unui fişier de tip COM cu acelaşi nume, pe acelaşi disc, însoţit de afişarea la consolă a lungimii programului şi fişierului COM creat.

Fişierele de tip COM conţin numai programul în cod binar, fără alte informaţii pentru sistem (cum ar fi adresa de încărcare, adresa de lansare, lungimea ş.a). Ca urmare, în sistemul CP/M toate programele COM se încarcă şi se execută în mod implicit de la adresa 100H (256 zecimal).

Dacă fişierul hexazecimal conţine un program care începe la o adresă mai mică ca 100H, programul LOAD semnalează această eroare (datorată probabil absenţei directivei ORG 100H).

Uneori poate fi necesară trecerea inversă, de la formatul COM la formatul HEX. Această transformare se poate face cu ajutorul programului utilitar GENHEX, program difuzat sub sistemul MP/M, dar fără să fie legat de acest sistem (se poate folosi sub CP/M).

Programul GENHEX se apelează prin comanda

GENHEX nume sau GENHEX d:nume

unde ,,nume'' este numele fişierului de tip COM.

5.1.2. Formatul hexazecimal al programelor obiect

Formatul hexazecimal (numit şi format Intel) a fost primul format de memorare a programelor 8080 pe un suport extern, suport care la început a fost banda perforată.

Utilizarea benzii perforate și a teleimprimatoarelor la primele microcalculatoare explică multe caracteristici ale formatului hex Intel care astăzi par greu de înțeles.

Menținerea formatului hexazecimal în multe sisteme de operare cu disc (SFDX-ISIS, CP/M, Monitor MADS) s-a făcut pentru compatibilitate și continuitate, dar și datorită unor avantaje ale acestui format:

— posibilitatea de a controla vizual direct programele în format HEX, prin folosirea exclusivă de caractere ASCII în acest format, prin delimitarea blocurilor de cîte 16 octeți cu caractere de trecere la linie nouă și prin însoțirea fiecărui bloc de adresa sa;

— posibilitatea de a încărca un program în format HEX la orice adresă de memorie;

— posibilitatea de verificare prin sumă de control pe fiecare bloc a operației de citire (importantă mai ales pe bandă de hîrtie, unde erorile de suport sau de citire sînt mai frecvente).

Dezavantajul principal al formatului HEX este lungimea mare a fișierelor HEX față de fișierele COM, datorată reprezentării fiecărui octet de memorie prin două caractere ASCII și informațiilor de control adăugate; astfel 16 octeți de memorie se reprezintă prin 16 octeți în format COM și prin 42 de octeți în format HEX.

Un fișier HEX este împărțit în mai multe blocuri sau înregistrări, de lungime variabilă, dar care nu poate depăși 16 octeți de date din memorie.

Fiecare înregistrare începe obligatoriu cu caracterul ':' (două puncte) și se termină cu o sumă de control pe un octet.

Între două înregistrări succesive pot fi oricîte caractere, de orice fel, cu excepția caracterului ':'; de obicei între blocuri se află caracterele CR, LF care asigură trecerea pe începutul liniei următoare la afișarea unei noi înregistrări.

Sfîrșitul unui fișier HEX este marcat printr-o înregistrare finală, care conține doar adresa de lansare în execuție a programului.

Toate înregistrările conțin la început lungimea înregistrării (între 1 și 16) și adresa de încărcare în memorie a datelor din înregistrarea respectivă; de asemenea, mai conțin un octet rezervat pentru tipul înregistrării și care este de obicei zero.

Așadar, formatul unei înregistrări oarecare, diferită de ultima, este următorul:

:LLAAAA00DD ... DDSS

Formatul ultimei înregistrări este:

:00AAAA01SS

unde:

LL este lungimea înregistrării (2 caractere)

AAAA este adresa de încărcare (4 caractere) sau adresa de execuție la ultima înregistrare;

DD octeți de date (cîte 2 caractere pentru fiecare octet);
SS suma de control (2 caractere).

Suma de control este calculată ca sumă modulo 256 negată a tuturor octeților din înregistrare (mai puțin ':'), astfel încît suma tuturor octeților, inclusiv suma de control, calculată pe un octet, să fie zero.

5.1.3. Utilizarea programelor ASM și MAC

Programele asamblor ASM și MAC diferă prin numărul de directive acceptate și prin lungime (8 koct pentru ASM și 12 koct pentru MAC), dar se folosesc în același mod.

Asamblorul ASM (sau MAC) prelucrează un fișier sursă de tipul ASM și produce trei fișiere cu același nume și pe același disc de unde s-a citit fișierul de intrare:

— fișierul de tip HEX conține codul obiect generat de asamblor, în format hexazecimal Intel;

— fișierul de tip PRN conține o listă de asamblare care va fi apoi imprimată sau afișată printr-o altă comandă;

— fișierul de tip SYM conține o tabelă de simboli, folosită de către programul depanator SID (sau ZSID) sau imprimată pentru utilizare la depanarea cu DDT.

În cursul asamblării se afișează la consolă liniile sursă cu erori, precedate de un cod de eroare.

Lista de asamblare (fișierul PRN) conține pentru fiecare linie sursă următoarele:

— un blanc, dacă linia este corectă sintactic sau o literă ce reprezintă un cod de eroare detectată de asamblor;

— adresa asociată instrucțiunii sau datelor din linia respectivă, în hexazecimal;

— codul obiect generat, în hexazecimal;

— forma sursă (simbolica) a instrucțiunii sau directivei.

Tabelul de simboluri (fișierul SYM) conține, în ordine alfabetică, toate numele simbolice folosite în program (etichete sau nume echivalate), împreună cu adresa sau valoarea asociată de către asamblor.

Comanda de asamblare cu generare de cod absolut hexa are forma următoare:

ASM nume   ASM d:nume

Extensia fișierelor sursă asamblate cu ASM și MAC este considerată implicit a fi „ASM", fiind ignorată orice extensie explicită după numele fișierului.

Dacă lipsește directiva ORG, atunci se consideră în mod implicit o directivă ORG 0000H, deci toate adresele se generează începînd de la zero.

Coduri de eroare la asamblare

D (Data error)
eroare de date în directive DB, DW;

E (Expression error)
expresie incorectă sau care nu poate fi calculată la asamblare;

L (Label error)
eroare de etichetă (posibil etichetă dublu definită);

N (Not implemented)
facilități neimplementate;

O (Overflow)
expresie prea complicată;

P (Phase error)
simbol cu valori diferite în cele două faze ale asamblării;

R (Register error)
nume de registru greșit;

V (Value error)
valoare greșită într-o expresie.

5.1.4. Limbajul de asamblare ASM/MAC

Un program sursă în limbaj de asamblare este format din mai multe linii sursă și se termină cu o linie care conține directiva END (liniile de după END sînt ignorate).

Programele pot fi scrise atît cu litere mari, cît și cu litere mici, deoarece literele mici sînt echivalente cu cele mari pentru asamblor (cu excepția celor din constantele alfanumerice).

Liniile sursă pot fi precedate de un număr de linie, iar terminatorul de linie este perechea de caractere ⟨CR⟩, ⟨LF⟩ sau caracterul „!". Este deci posibil să se scrie mai multe instrucțiuni pe o linie terminată cu ⟨CR⟩ ⟨LF⟩. Exemplu:

RLC! RLC! RLC! RLC

O linie sursă are în general patru zone:

— zona etichetă (opțională);
— zona de cod operație (obligatorie);
— zona de operanzi (funcție de codul operației);
— zona de comentariu (opțională).

Zona etichetă conține un nume simbolic urmat de un spațiu sau de caracterul „:" ca terminator de etichetă.

Numele simbolice utilizate ca etichete pot avea maximum 16 caractere și pot conține caracterul „$" (care este însă ignorat de asamblor).

Zona de cod operație conține numele simbolic al unei instrucțiuni mașină sau numele unei directive de asamblare sau numele unei macroinstrucțiuni (numai la MAC).

Zona de operanzi conține în general expresii formate din operanzi și operatori. Operanzii pot fi:

— etichete simbolice;
— constante numerice;
— constante șir de caractere;
— cuvinte rezervate (nume de registre ș.a.);
— coduri mnemonice de instrucțiuni;
— caracterul „$" interpretat ca „adresă curentă".

Comentariile pot fi precedate de caracterul „;" sau „*". O linie care începe cu caracterul „;" sau „*" în prima poziție este considerată ca linie comentariu și nu este analizată la fel ca celelalte linii sursă.

Constantele numerice pot fi scrise în binar (terminate cu „B"), octal (terminate cu „O" sau „Q"), hexazecimal (terminate cu „H") sau zecimal (terminate eventual cu „D").

De observat că pentru a deosebi constantele hexazecimale care încep cu una din cifrele A ... F de numele simbolice este necesar ca aceste constante să fie precedate de o cifră zero; de exemplu, se va scrie 0C0H și nu C0H.

Numele de registre și codurile numerice de instrucțiuni sînt *nume rezervate* și nu pot fi utilizate ca etichete simbolice (între numele de registre se includ și M, SP, PSW).

Caracterul „$" în zona operand desemnează adresa curentă a instrucțiunii în care apare.

O constantă șir este un șir de caractere (maximum 64 caractere) incluse între caracterele apostrof ('). Caracterul apostrof (') poate fi inclus în șir dacă este dublat (se introduce de două ori același caracter).

Expresiile din zona operand pot conține următorii operatori:

— operatorii unari: +, —
— operatorii aritmetici: +, —, *, /, MOD
— operatorii logici: NOT, AND, OR, XOR
— operatorii de deplasare: SHL, SHR

Expresia x MOD y reprezintă restul împărțirii lui x la y.

Expresia x SHL y are ca rezultat valoarea lui x deplasată la stînga de y ori (cu introducere de zerouri).

Expresia x SHR y are ca rezultat valoarea lui x deplasată la dreapta de y ori (cu introducere de zerouri).

Este posibilă utilizarea de subexpresii în paranteze ca operand într-o expresie.

Valoarea calculată a unei expresii poate fi de un octet sau de un cuvînt, în funcție de codul instrucțiunii; folosirea de valori mai mari de 255 în instrucțiunile pe octet este semnalată ca eroare.

Directive de asamblare ASM

DB exp[,exp...]
DB 'text'[,'text'...]

Directiva DB („Define Byte") generează una sau mai multe constante de cîte un octet fiecare, sau șirurile de caractere date ca operanzi.

DW exp[,exp...]

Directiva DW („Define Word") generează una sau mai multe constante de cîte un cuvînt fiecare (un cuvînt are doi octeți). Constantele pot fi și alfanumerice, de unul sau două caractere între ghilimele simple, dar nu șiruri de caractere.

DS exp

Directiva DS („Define Storage") rezervă o zonă de memorie de lungime egală cu valoarea expresiei (lungimea în octeți).

END exp

Directiva END marchează sfîrșitul programului sursă asamblat și, eventual, conține adresa de lansare în execuție. Directiva END poate lipsi, fără să afecteze asamblarea.

nume **EQU** exp

Directiva EQU („Equivalence") echivalează numele din zona etichetă cu valoarea expresiei din zona operand (de obicei atribuie un nume unei valori constante).

ORG exp

Directiva ORG („Origin") stabilește adresa de încărcare, pentru instrucțiunile sau datele care urmează, fiind egală cu valoarea expresiei operand. Sînt permise mai multe directive ORG într-un program, fără să se verifice ordinea adreselor sau suprapunerea unor zone de memorie între ele. Deoarece programele generate sînt absolute, este necesară cel puțin o directivă ORG la începutul programului.

nume **SET** exp

Directiva SET este echivalentă cu directiva EQU, dar poate fi utilizată de mai multe ori cu același nume, pentru redefinirea valorii asociate acestui nume. Se folosește de obicei împreună cu directivele de asamblare condiționată.

IF exp
ENDIF ...

Directiva de asamblare condiționată **IF** testează valoarea expresiei „exp" și în funcție de această valoare include (valoarea nenulă) sau nu (valoarea zero) în asamblare secvența de instrucțiuni și (sau) directive cuprinsă între IF și ENDIF.

Directive de asamblare MAC

Macroasamblorul MAC acceptă toate directivele asamblorului ASM, plus următoarele directive necesare definirii și utilizării de macroinstrucțiuni:

nume **MACRO** arg1,arg2,...

Directiva MACRO trebuie să înceapă orice macrodefiniție; ea specifică numele și argumentele formale ale macroinstrucțiunii.

ENDM

Directiva ENDM termină o macrodefiniție. Macroinstrucțiunile trebuie definite înainte de a fi utilizate.

MACLIB fișier[fișier...]

Directiva MACLIB declară numele bibliotecii (bibliotecilor) de macroinstrucțiuni utilizate, dar definite în afara programului. Fișierul bibliotecă are tipul implicit „LIB".

TITLE 'text'

Directiva TITLE stabilește un titlu care se imprimă pe fiecare pagină a listei de asamblare.

PAGE [n]

Directiva PAGE stabilește numărul de linii pe pagină la valoarea „n" sau comandă trecerea la pagină nouă.

5.2. Programe asamblor cu generare de cod obiect relocabil

Utilizarea unui asamblor de programe relocabile poate fi necesară în următoarele situații:

— programul este foarte mare și este dezvoltat pe module, asamblate și testate separat;

— programul utilizează subprograme dintr-o bibliotecă;

— modulele (subprogramele) în limbaj de asamblare trebuie legate cu alte module obiect scrise într-un limbaj evoluat;

— programul se mută, după încărcare, la adrese mari și se execută la aceste adrese;

— sînt necesare anumite directive de asamblare sau alte facilități oferite numai de asamblorul M80;

— programele se scriu cu mnemonice de Z80 și deci pot fi asamblate numai cu M80.

Deoarece linkeditorul L80 generează, la cerere, un fișier tabelă de simboli globali (de tip SYM), este posibilă și depanarea simbolică a programelor asamblate cu M80.

În cazul programelor relocabile se utilizează noțiunea de modul de program (sau modul obiect); un modul este un text sursă terminat cu directiva END și care poate fi asamblat separat de celelalte module din componența unui program.

Un fișier sursă conține un singur modul sursă, din care asamblorul produce un modul obiect.

Legarea mai multor module obiect într-un singur program executabil se face după asamblare, în faza de editare a legăturilor (sau linkeditare). Faza de linkeditare este necesară întotdeauna după o asamblare cu cod relocabil, chiar și pentru programele formate dintr-un singur modul.

5.2.1. Cod obiect relocabil. Formatul Microsoft

Caracteristic pentru programele (modulele) relocabile este faptul că la asamblarea lor nu se cunosc adresele la care vor fi încărcate și executate, deoarece vor fi legate împreună cu alte module și numai atunci vor primi adrese definitive.

Un asamblor relocabil generează, pentru instrucțiunile cu referire directă la memorie, adrese provizorii, modificabile (relocabile, translatabile) sub forma unor adrese relative la o bază.

În cazul cel mai simplu există o singură adresă de bază pentru un modul obiect, adresă considerată egală cu zero la asamblare; toate adresele instrucțiunilor și datelor din modulul respectiv se generează relativ la baza modulului.

Asamblorul M80 poate utiliza mai multe adrese de bază în cadrul unui singur modul și deci pot exista mai multe tipuri de adrese relocabile:

— adrese relocabile în cadrul secțiunii de cod (numită și segment de program);

— adrese relocabile în cadrul secțiunii de date (numită și segment de date);

— adrese relocabile în cadrul unei secțiuni comune (numită și bloc de date comune).

Într-un modul obiect Microsoft poate exista o singură secțiune de program, o singură secțiune de date și mai multe secțiuni de comun, fiecare avînd un nume.

Adresele relocabile sînt modificate (relocate) de către linkeditor, prin adăugarea adresei de bază stabilite pentru fiecare modul și secțiune cu adresele relative furnizate de asamblor.

În afara adreselor de memorie relocabile, un modul obiect relocabil poate conține și adrese de simboluri externe, din alte module. Asamblorul relocabil înlocuiește aceste referințe externe cu zero și transmite linkeditorului informațiile necesare completării acestor adrese.

Etichetele simbolice globale sînt de două tipuri: referințe externe (folosite în modulul curent dar definite în alte module) și puncte de intrare (definite în modulul curent, folosite și în alte module); prin asemenea etichete se realizează referințele simbolice între module.

Linkeditorul atribuie adrese absolute simbolurilor globale și completează instrucțiunile care conțin referiri externe.

Formatul obiect relocabil Microsoft

Este un format foarte compact, dar mai greu de citit, deoarece înregistrările sînt șiruri de biți de lungime variabilă, nedivizibilă prin 8 în general. Ca urmare, un octet poate conține biți din înregistrări diferite sau din cîmpuri diferite ale unei înregistrări.

Primul bit din fiecare înregistrare deosebește înregistrările cu text obiect absolut (bit 0) de înregistrările cu text obiect ce trebuie prelucrat de linkeditor (bit 1).

O înregistrare cu date absolute are o lungime de 9 biți: primul bit indică tipul (0 = date absolute), iar următorii 8 biți formează un octet de date care poate reprezenta codul unei instrucțiuni sau un octet dintr-o constantă.

Octeții din textul obiect absolut sînt încărcați în memorie ca atare fără nici o prelucrare.

Înregistrările care au primul bit 1 pot fi de mai multe tipuri și pot avea lungimi diferite.

Următorii doi biți din înregistrările care încep cu 1 codifică tipul înregistrării, după cum urmează:

00 înregistrări de control;
01 înregistrări cu adrese relative la secțiunea de program;
10 înregistrări cu adrese relative la secțiunea de date;
11 înregistrări cu adrese relative la secțiunea de comun.

Înregistrările care conțin adrese relative au o lungime de 19 biți: primii 3 biți codifică tipul înregistrării (101, 110, 111), iar următorii 16 biți reprezintă valoarea adresei relative.

Înregistrările de control (de tipul 100) sînt la rîndul lor de mai multe feluri, iar tipul înregistrării de control este codificat prin următorii 4 biți din înregistrare.

După primii 7 biți de identificare o înregistrare de control poate conține:

— un cîmp A de 18 biți reprezentînd o adresă;

— un cîmp B avînd între 11 și 59 de biți, reprezentînd un nume simbolic cu o lungime de la 1 la 7 caractere (dintre care numai primele 6 sînt efectiv utilizate de linkeditor);

— un cîmp A și un cîmp B (o adresă și un nume);

— nici cîmpul A, nici cîmpul B (ultima înregistrare din fișier).

Cîmpul A are în primii 2 biți tipul adresei codificat la fel ca la înregistrările cu adrese relative.

Cîmpul B conține în primii 3 biți numărul de caractere din numele simbolic, după care urmează imediat primul caracter (fiecare caracter ocupă 8 biți).

Exemple de înregistrări care conțin numai cîmpul A:

— adresă relativă simbol extern;
— lungime secțiune de date;
— lungime secțiune de program;
— adresă de încărcare.

Exemple de înregistrări care conțin numai cîmpul B:

— nume de punct de intrare;
— nume de bloc comun;

— nume modul obiect;
— nume bibliotecă.

Exemple de înregistrări care conţin cîmpurile A şi B:
— lungime bloc comun cu nume dat;
— adresă asociată unui punct de intrare;
— adresa lanţului de referiri la un simbol extern.

5.2.2. Utilizarea asamblorului Macro-80

Sînt posibile două moduri de utilizare a asamblorului M80:

— pentru asamblarea unui singur program (modul) sursă se introduce o singură linie de comandă de forma:

M80 frel, fprn = fmac/opt 1/opt 2/...

— pentru asamblarea mai multor programe (module) sursă se încarcă asamblorul prin comanda:

M80

şi apoi se pot introduce mai multe linii de forma:

frel, fprn = fmac/opt 1/opt 2/...

ca răspuns la caracterul „*", ce arată că asamblorul este gata să primească o nouă comandă.

Parametrii liniei de comandă au următoarea semnificaţie:

frel — fişier obiect relocabil produs de asamblor (are tipul implicit „REL");

fprn — fişier de listare produs de asamblor (are tipul implicit „PRN", pe disc);

fmac — fişier sursă citit de M80 (are tipul implicit „MAC")

Numele de fişiere fmac, frel, fprn urmează formatul general:

d: nume. tip   sau   nume

Dispozitivul suport „d" poate lipsi cînd se referă la discul implicit, iar tipul fişierelor poate lipsi, dacă se respectă convenţiile implicite (fişierul sursă poate avea orice extensie, dar trebuie scrisă explicit) Discul suport al fişierului de intrare este preluat implicit şi pentru fişierele de ieşire.

Opţiunile de asamblare „opt" sînt:

/C generează referinţe încrucişate pe lista de asamblare;
/L produce fişier de listare;
/H afişarea în hexa a memoriei pe lista de asamblare;
/O afişarea în octal a memoriei pe lista de asamblare;
/R generează fişier obiect;
/Z generează cod obiect pentru Z80.

Ca fişier de listare se pot utiliza următoarele nume de dispozitive periferice:

„TTY:" pentru consola sistem;
„LST:" pentru imprimantă.

Fişierele „frel" şi „fprn" sînt opţionale; de asemenea, tipurile fişierelor sursă şi obiect pot fi omise în comandă (dacă fişierele sursă sînt de tip „MAC").

La consolă se afişează întotdeauna liniile cu erori, chiar dacă nu se cere lista de asamblare.

Exemple de comenzi de asamblare:

1) M80 B: TESTM, TTY: = B: TESTM

Se asamblează programul din fişierul B: TESTM.MAC, se generează fişierul B: TESTM.REL şi se listează la consolă codul sursă.

2) M80 , = TEST.ASM

Se face o verificare sintactică a programului din fişierul TEST.ASM, fără a genera fişier obiect (erorile sînt afişate).

3) M80 = B: TEST/R/L

Se asamblează fişierul B: TEST.MAC şi se produc două fişiere pe acelaşi disc: TEST.REL şi TEST.PRN.

Această comandă este echivalentă cu comanda:

M80 B: TEST, B: TEST = B: TEST

sau, ma complet:

M80 B: TEST.REL, B: TEST.PRN = B: TEST.MAC

4) M80
*=P1/R
*=P2/R
*CTRL/C

Se asamblează succesiv fişierele P1.MAC şi P2.MAC generînd fişierele P1.REL şi P2.REL. Ieşirea din asamblor se face cu CTRL/C.

Convenţii utilizate în lista de asamblare

Formatul unei linii din lista de asamblare este **următorul**:

[crf] [err] loc m xx xxxx sursa

unde:

crf	număr de referinţă încrucişată (opţional);
err	cod de eroare (o literă);
loc	adresa locaţiei de memorie;
m	indicator mod de adresare;
xx...	conţinutul locaţiei de memorie.
sursa	linia sursă (instrucţiune sau directivă)

Indicatorul de mod „m" poate avea valorile următoare:

spaţiu adresă absolută;
' adresă relativă la segmentul de cod;
" adresă relativă la segmentul de date;
! adresă relativă în blocul comun;
* referinţă externă.

În tabelul de simboluri se folosesc următoarele notaţii:

* simbol extern;
I simbol public (punct de intrare);

C nume bloc comun;

U simbol nedefinit.

O listă de *referinţe încrucişate* este o listă în care fiecare nume simbolic din program este însoţit de numerele liniilor în care apare (linia în care este definit e marcată prin caracterul ,#'). O asemenea listă se poate obţine într-un fişier de tip LST mai special, care are şi liniile precedate de un număr de referinţă încrucişată, folosind un program separat numit CREF80 („cross-reference program") în felul următor:

CREF80

*TEST = TEST

sau

* = TEST

în care TEST este numele unui fişier TEST.CRF creat de M80 ca urmare a folosirii opţiunii /C. Deci, programul CREF80 citeşte un fişier de tip CRF şi produce un fişier de tip LST.

Coduri de eroare la asamblare

A (Argument error)
argument de directivă greşit;

C (Conditional nesting error)
directive condiţionale incluse greşit;

D (Double defined symbol)
simbol dublu definit;

E (External error)
eroare la o referinţă externă;

M (Multiple defined symbol)
simbol multiplu definit;

N (Number error)
constantă numerică greşită;

O (Opcod error)
cod operaţie greşit sau altă eroare de sintaxă;

P (Phase error)
eroare de fază la asamblare;

Q (Questionable syntax)
linie terminată incorect;

R (Relocation error)
utilizare incorectă a unui simbol relocabil într-o expresie;

U (Undefined symbol)
simbol nedefinit;

V (Value error)
valoare greşită a unei constante.

5.2.3. Limbajul de asamblare Macro-80

Limbajul de asamblare acceptat de M80 este compatibil cu limbajele acceptate de alte asambloare pentru 8080, dar prezintă şi unele particularităţi ce sînt prezentate în continuare.

Particularităţile de sintaxă M80 faţă de ASM/MAC:

— o linie sursă poate avea cel mult 132 de caractere şi poate fi precedată de un număr de linie, cu condiţia ca fiecare cifră din acest număr să aibă bitul 7 egal cu 1;

— literele mici sînt acceptate numai în comentarii şi în constante alfanumerice;

— în cadrul etichetelor simbolice sînt admise următoarele caractere speciale, tratate ca litere: „$",. „.", „?", „@", „—";

— sînt admise constante sau expresii în zona de cod mnemonic, ele fiind considerate ca operanzi ai unei directive DB implicite:

```
zero: 0
```

este echivalentă cu:

```
zero: DB 0
```

— o etichetă urmată de două caractere „:" este tratată ca simbol public (punct de intrare):

```
SUBR: : PUSH     H
```

este echivalentă cu

```
PUBLIC    SUBR
     ...
SUBR:     PUSH     H
```

— un simbol urmat de două caractere „#" este tratat ca simbol extern:

```
CALL      MUL##
```

este echivalent cu:

```
EXTRN     MUL
     ...
CALL      MUL
```

— constantele numerice pot fi binare (sufix B), octale (sufix O sau Q), hexazecimale (sufix H sau prefix X), zecimale (sufix D sau nici un sufix);

— constantele numerice prea mari sînt automat trunchiate la un cuvînt (16 biţi);

— şirurile de caractere pot fi delimitate prin ghilimele simple (apostrof) sau duble;

— Expresiile din zona operand pot conţine:

operatorii aritmetici +, —, *, / şi semnul minus;
operatorii simbolici LOW, HIGH, MOD, SHR, SHL;
operatorii de relaţie EQ, NE, LT, LE, GT,GE;
operatorii logici NOT, AND, OR, XOR;
paranteze rotunde.

— operatorul LOW aplicat asupra unui cuvînt are ca rezultat octetul inferior din acel cuvînt, iar operatorul HIGH are ca rezultat octetul superior al unui cuvînt;

— operatorii simbolici, de relație și logici trebuie separați de operanzi prin spații.

— se pot utiliza ca operanzi de un octet coduri simbolice de instrucțiuni:

MVI A, (JMP)
MVI C, MOV A, B

— simbolurile sînt de 4 tipuri:
absolute;
relative la segmentul de date;
relative la segmentul de program;
relative la blocul de comun;

Expresiile permise cu aceste simboluri sînt:
⟨oarecare⟩ + ⟨absolut⟩ —⟩ ⟨oarecare⟩
⟨oarecare⟩ — ⟨absolut⟩ —⟩ ⟨oarecare⟩
⟨oarecare⟩ — ⟨oarecare⟩ —⟩ ⟨absolut⟩

— simbolurile externe pot fi folosite în expresii aritmetice cu operatorii „+", „—".

Directivele asamblorului M 80

ASEG

Început de segment absolut. Această directivă este echivalentă cu directiva CSEG însoțită de opțiunea /P la linkeditare.

COMMON //
COMMON /nume/

Început de bloc de date comun.

CSEG

Începutul unui segment de cod. Directiva CSEG este implicită pentru orice programe, deci poate lipsi în general. Ea este necesară atunci cînd segmentul de cod este precedat de un bloc comun, pentru delimitarea celor două segmente.

DB exp[, exp ...]
DB "sir"[, "sir" ...]

Generează una sau mai multe constante de un octet, corespunzătoare expresiilor sau șirurilor de caractere din zona operand.

DC "sir"

Generează un șir de caractere ASCII și pune 1 în bitul cel mai semnificativ (bitul 7) din ultimul caracter (pentru detectarea sfîrșitului șirului de caractere).

DS exp

Rezervă o zonă de memorie de lungime egală cu valoarea expresiei ⟨exp⟩

DSEG

Început de segment de date.

DW exp[, exp ...]
Generează una sau mai multe constante de un cuvînt.

END [exp]
Marchează sfîrşitul unui program şi, eventual indică adresa de lansare în execuţie.

ENTRY nume [, nume ...]
PUBLIC nume [,nume ...]
Aceste două directive echivalente declară unul sau mai multe simboluri publice (puncte de intrare), care pot fi folosite şi din alte module asamblate separat.

nume **EQU** exp
Atribuie numelui din zona etichetă valoarea expresiei din zona operand.

EXTRN nume [, nume ...]
EXT nume [, nume ...]
Aceste două directive echivalente declară unul sau mai multe simboluri externe, definite în alte module, dar folosite în modulul curent.

NAME ('nume')
Atribuie un nume modulului curent (echivalentă cu TITLE).

ORG exp
Stabileşte adresa de implantare pentru codul sau zonele de date ce urmează.

PAGE [exp]
Produce trecerea la o pagină nouă în cursul listei de asamblare. O pagină are implicit lungimea de 50 de linii sau lungimea dată de operandul directivei (10—255).

nume **SET** exp
Atribuie numelui din stînga valoarea expresiei din dreapta, ca şi directiva EQU, dar permite redefinirea unui nume de mai multe ori în program.

SUBTTL "text"
Generează un subtitlu, de maximum 60 de caractere, imprimat automat pe fiecare pagină de listare.

TITLE "text"
Generează un titlu, imprimat pe prima linie din fiecare pagină a listei de asamblare; primele 6 caractere se consideră drept nume al modulului obiect generat.

.COMMEN /text/
Reprezintă o altă posibilitate de a scrie comentarii în textul sursă; ca delimitatori de început şi de sfîrşit comentariu se poate folosi orice caracter (nu neapărat caracterul „/").

.PRINTX/text/
Afişează la consolă textul respectiv în momentul asamblării acestei directive. Textul poate fi încadrat de orice caractere identice.

.RADIX exp
Modifică baza de numeraţie implicită pentru constantele numerice,

.REQUEST fisier[, fisier ...]
Cere linkeditorului să caute în bibliotecile indicate pentru rezolvarea referințelor externe întîlnite în program.

.XLIST / .LIST
Oprire (reluare) listare text sursă în fișierul de listare.

nume **MACRO** arg1, arg2, ...
Început de macroinstrucțiune.

ENDM
Sfîrșit de macroinstrucțiune.

EXITM
Termină expandarea unei macroinstrucțiuni.

LOCAL arg1, arg2, ...
Creează cîte un simbol unic pentru fiecare dintre argumentele formale conținute în cursul expandării unei macroinstrucțiuni. Etichetele astfel generate au forma ..nnnn unde „n" este o cifră zecimală.

.LALL
Listează tot textul macro la toate expandările.

.SALL
Listează numai codul obiect produs la o macroinstrucțiune.

.XALL
Interzice listarea macro-expandărilor.

Caracterul „&" se folosește pentru concatenare de texte sau simboluri în cursul expandării unei macroinstrucțiuni. Un argument formal introdus între "" nu este substituit decît dacă este precedat de „&".

Caracterul „!" utilizat înaintea unui caracter într-un argument face ca acest caracter să fie tratat ca un literal (să fie copiat întocmai și nu substituit).

Parantezele ascuțite („⟨ ⟩") se folosesc tot pentru literali în cadrul argumentelor. Notațiile „!" și „⟨;⟩" sînt echivalente.

Comentariile dintr-o macrodefiniție care sînt precedate de „;;" nu sînt memorate și deci nu mai apar la expandarea macroinstrucțiunii.

Directive de asamblare condiționată

IF NUL arg
Directiva condițională are un rezultat fals, dacă, în timpul expandării, primul caracter din argument este orice altceva decît „;" sau ⟨CR⟩.

```
IFxx arg
    ...
[CELSE
    ...]
ENDIF
```

unde

IFxx poate fi una dintre următoarele directive:

IF/IFT exp — adevărat, dacă ⟨exp⟩ este diferită de zero;
IFE/IFF exp — adevărat, dacă expresia ⟨exp⟩ este zero;

IF1 adevărat pentru prima trecere a asamblării;
IF2 adevărat pentru a doua trecere a asamblării;
IFNDEF simbol adevărat, dacă ⟨simbol⟩ este nedefinit;
IFDEF simbol adevărat, dacă ⟨simbol⟩ este definit sau este declarat extern;
IFB ⟨arg⟩ adevărat, dacă argumentul este blanc („ ”) (parantezele unghiulare sînt necesare);
IFNB ⟨arg⟩ adevărat, dacă argumentul este diferit de blanc.

Este permisă includerea directivelor condiționale pe orice număr de nivele.

Argumentele directivelor condiționale trebuie să fie cunoscute din prima trecere a asamblării, deci să fie definite înainte de a fi utilizate ca argumente.

Directive de asamblare repetată

REPT exp

Repetă liniile care urmează pînă la o directivă ENDM de cîte ori arată valoarea expresiei „exp” (valoare pe 16 biți).

IRPC arg, ”sir”

Repetă instrucțiunile care urmează pînă la o directivă ENDM de atîtea ori cîte caractere conține șirul dat, înlocuind de fiecare dată argumentul formal cu caracterul următor din șir.

IRP arg,⟨arg1, arg2,. . .⟩

Repetă liniile care urmează pînă la o directivă ENDM de atîtea ori cîte argumente sînt în lista dintre parantezele unghiulare, substituind de fiecare dată argumentul formal cu argumentul următor din listă.

5.3. Editorul de legături Link-80 și bibliotecarul Lib-80

Programul L80 este un editor de legături pentru module obiect relocabile în format „REL”, rezultate din compilare FORTRAN (cu F80), din asamblare cu asamblorul M80, din compilare BASIC (cu BASCOM) sau din compilare COBOL.

Linkeditorul L80 poate lega împreună mai multe module obiect aflate în fișiere de tip „REL” cu module extrase din una sau mai multe biblioteci de subrutine relocabile, producînd un singur program executabil care se încarcă în memorie și care, la cerere, este scris într-un fișier de tip .COM.

Bibliotecile prelucrate de L80 trebuie create cu un program bibliotecar special numit LIB80.

Listarea numelor modulelor dintr-o bibliotecă se poate face cu bibliotecarul sau chiar cu linkeditorul L80, dacă se leagă între ele toate modulele din bibliotecă și se folosește opțiunea /M de afișare a simbolurilor globale.

Linkeditorul L80 caută implicit într-o bibliotecă cu numele FORLIB. REL (necesară pentru modulele generate de compilatorul FORTRAN), dar se poate cere explicit căutarea în alte biblioteci pentru satisfacerea referințelor externe (opțiunea /S).

Nu este prevăzută crearea de programe segmentate cu ajutorul linkeditorului L80.

5.3.1. Utilizarea programului Link-80

Comanda de linkeditare poate avea una dintre formele:

L80 frel1/opt, frel2/opt,.../E

sau

L80 frel1/opt, frel2/opt,..., fcom/N/E;

cînd se dorește obținerea unui fișier obiect de tip .COM,

sau

```
L80
*frel1/opt
*frel2/opt
*/E
```

sau

```
L80
*frel1/opt
*frel2/opt
 ...
*/E, fcom/N
```

cînd se dorește obținerea unui fișier „fcom" de tip .COM (ordinea opțiunilor /E și /N poate fi oarecare, dar /N este o opțiune asociată fișierului de ieșire și care îl deosebește de fișierele de intrare. În lipsa opțiunii /N, linkeditorul încarcă în memorie programul creat și nu creează un fișier disc cu acest program.

Sfîrșitul introducerii de module obiect și de opțiuni este marcat prin opțiunea /E (Exit) sau /G (Go), moment în care linkeditorul caută în biblioteca implicită și (sau) în bibliotecile date explicit pentru a rezolva referințele externe rămase și pentru a încheia editarea.

Opțiunile „opt" pot fi:

/D: adresa

Stabilire adresa de încărcare a segmentelor de date și de comun; adresa implicită a segmentului de date este 103H.

/E
/E: nume

Terminare listă de fișiere de intrare și începere căutare în biblioteci; eventual este însoțită de indicarea numelui simbolic global ce reprezintă adresa de lansare.

/G
/G: nume
Lansare automată în execuție după linkeditare, eventual cu indicarea numelui adresei de lansare (simbol public).
/M
Listare început și sfîrșit segment de date și de program, precum și numele simbolurilor globale (publice și externe).
/P: adresa
Stabilire adresă de încărcare a segmentului de cod din următorul modul obiect prelucrat (nu are efect asupra modulului curent). Adresa implicită a segmentului de cod este 100H.
/R
Relansarea de la început a linkeditorului, fără reîncărcarea sa în memorie (se folosește după o eroare la numele fișierelor de intrare).
fișier/S
Se cere linkeditorului să caute în biblioteca indicată prin ⟨fișier⟩, pentru rezolvarea referințelor externe.
/U
Listare început și sfîrșit segment de date și de program, precum și numele simbolurilor globale nedefinite.
/Y
Generare fișier cu extensia „SYM" care conține tabela simbolurilor globale din modulele legate împreună.

Observații:

— Opțiunile /D și /P pot lipsi; în acest caz linkeditorul plasează segmentul de date înaintea segmentului de cod la fiecare modul și introduce o instrucțiune de salt (JMP) peste segmentul de date. Dacă lipsește segmentul de date, atunci segmentul de program urmează imediat instrucțiunii JMP.

— Adresa de încărcare implicită este 100H.

— Dacă se folosesc directive ORG într-un program, atunci adresele conținute în aceste directive sînt tratate ca adrese relative la baza de încărcare implicită (100H) și nu ca adrese absolute.

— Adresa primei locații libere după segmentul de date (sau după segmentul de cod dacă lipsește segmentul de date) este introdusă de linkeditor în simbolul global $MEMRY, dacă a fost definit.

Exemple de comenzi de linkeditare.

1) L80 TEST, TEST/N/E

Se face relocarea programului obiect citit din fișierul TEST.REL, se creează un fișier TEST.COM și se revine în sistem.

Se poate scrie și astfel:

L80 TEST/E, TEST/N

2) L80 TEST/G

Se face relocarea programului obiect citit din fișierul TEST.REL, se încarcă programul absolut în memorie și se lansează în execuție.

ß) L80 G1,G2
*GLIB/S
*GRAF/N/E

Se leagă împreună modulele citite din fişierele G1.REL şi G2.REL cu subrutinele apelate din aceste module şi extrase din biblioteca GLIB; se crează fişierul GRAF.COM.

4) L80
*G1
*G2
*GLIB/S
*GRAF/N
*/E

Acelaşi efect cu secvenţa din exemplul precedent.

5.3.2. Utilizarea programului Lib-80

Fişierul ce conţine bibliotecarul se numeşte LIB80.COM sau LIB.. COM.

Programul bibliotecar LIB—80 realizează următoarele funcţii:

— crearea de biblioteci relocabile pornind de la fişiere rezultate din compilări şi asamblări diferite;

— listarea modulelor dintr-o bibliotecă împreună cu toate numele de simboluri globale;

— extragerea de module obiect din fişiere sau biblioteci în alte fişiere relocabile.

Apelarea bibliotecarului se face prin comanda:

LIB sau LIB80

După încărcarea sa în memorie, LIB80 afişează un asterisc („*") şi aşteaptă comenzi.

Formele unei comenzi LIB-80 pot fi:

*flib = frel1, frel2,.../opt

sau:

*flib/opt

sau:

*flib = frel1
*frel2
*frel3
...
*/E

Numele fişierelor de intrare se pot da fie în aceeaşi linie, separate prin virgule, fie pe linii separate.
„flib" este numele fişierului bibliotecă creat (poate lipsi).
„frel" este numele unui fişier de intrare de tip „REL".
„opt" sînt opţiuni pentru bibliotecar.

cd 918 c.8

Un fişier relocabil poate conţine unul sau mai multe module obiect. Un modul obiect corespunde unei unităţi de program Fortran (program principal, subrutina sau funcţie) sau unei secvenţe în limbaj de asamblare ce conţine directive ENTRY şi eventual alte directive pentru stabilire nume modul (NAME, TITLE).

Bibliotecarul LIB-80 concatenează module obiect şi nu fişiere relocabile.

Pentru un fişier relocabil care conţine mai multe module obiect se poate preciza numele modulului sau modulelor care se extrag din fişierul respectiv. Dacă nu se indică în mod explicit care module se extrag atunci se consideră implicit toate modulele din fişierul respectiv.

Specificarea unui singur modul „MOD" din fişierul „FILE":

FILE⟨MOD⟩ sau FILE⟨MOD + n⟩ sau FILE⟨MOD — n⟩

modulul MOD sau modulul „n" după sau înainte de MOD

Specificarea prin nume a fiecărui modul extras:

FILE⟨MOD1, MOD2, MOD3⟩

modulele MOD1, MOD2, MOD3

Specificarea unei liste de module adiacente:

FILE⟨MOD1..MOD5⟩

toate modulele între MOD1 şi MOD5 inclusiv MOD1, MOD5.

FILE⟨..MOD5⟩

toate modulele de la începutul lui FILE pînă la MOD5 şi MOD5.

FILE⟨MOD1..⟩

toate modulele care urmează lui MOD1 în FILE, inclusiv MOD1.

Dacă nu se specifică explicit fişierul de ieşire, atunci se consideră implicit ca fişier de ieşire biblioteca FORLIB.REL.

Opţiuni LIB80:

/L

Listarea modulelor dintr-un fişier sau dintr-o bibliotecă cu punctele de intrare şi referinţele externe din fiecare modul.

/E

Ieşire din LIB80 în CP/M. Biblioteca creată primeşte tipul .REL, deoarece pe parcursul creării are tipul .LIB.

/R

Modifică tipul bibliotecii create din .LIB în .REL; nu se iese.

/U

Listare referinţe nerezolvate (referinţe înapoi) după parcugerea fişierului specificat

/C

Şterge biblioteca în curs de creare; reluare LIB-80 de la început

Exemple de utilizare LIB-80:

1) Crearea unei noi biblioteci GLIB.REL pe baza modulelor din fişierele MOVE.REL, DRAW.REL (provenite din Fortran) şi DOT.REL (provenit din asamblare). Subrutina DOT este apelată din MOVE şi din DRAW; DRAW apelează pe MOVE.

```
A>LIB
*GLIB = DRAW, MOVE, DOT
*/E
```

2) Adăugarea modulelor din fișierul DASH.REL la biblioteca GLIB:

```
A>LIB
*TEMP = GLIB, DASH
*TEMP/R
*GLIB = TEMP/E
```

3) Verificarea ordinii corecte a modulelor în biblioteca GLIB:

```
A>LIB
*GLIB/U
*GLIB/L/E
```

4) Extragerea modulelor MOVA și DRAWA din biblioteca GLIB în fișierul GRAF.REL:

```
A>LIB
*GRAF = GLIB<MOVEA, DRAWA>/E
```

Observații

— Extragerea unor module obiect dintr-o bibliotecă sau dintr-un fișier nu șterge aceste module din fișier; nu se poate face o ștergere selectivă de module dintr-un fișier relocabil.

— Dacă există anterior pe disc un fișier cu numele bibliotecii create atunci se șterge acest fișier fără nici o avertizare.

— adăugarea de noi module se face întotdeauna la sfîrșitul fișierului destinație; nu se pot intercala noi module între alte module dintr-o bibliotecă.

— Nu se poate da o comandă de forma:

```
GLIB = GLIB, DASH
```

cu intenția de adăugare a fișierului DASH.REL la GLIB.REL.

— Nu se recomandă modificarea bibliotecii standard FORLIB.REL

5.4. Programe pentru depanare de cod mașină

Pentru depanarea programelor scrise în limbaj de asamblar,e în sistemul CP/M se poate utiliza unul din următoarele trei programe:

DDT (Dynamic Debugging Tool)
SID (Symbolic Interactive Debugger)
ZSID (Z80 Symbolic Interactive Debugger)

Principala diferență între programul DDT și, respectiv, programele SID, ZSID constă în modul în care se pot referi comenzile programului de depanare la adrese din programul depanat: în DDT singura posibilitate este ca adresele să fie referite prin numere hexazecimale, în SID și ZSID este posibilă și o referire simbolică, prin numele etichetei din program, precedată de un punct.

Etichetele simbolice din program sînt transmise de asamblor programului depanator prin fişierul de tip „SYM".

Programul ZSID este destinat maşinilor CP/M cu procesor Z80.

Diferenţele dintre programele SID şi ZSID apar la comenzile de asamblare-dezasamblare, care folosesc alte coduri mnemonice, şi la comanda de afişare a registrelor.

5.4.1. Utilizarea programelor DDT şi SID

Programele DDT şi SID pot fi folosite nu numai pentru depanarea unor programe obiect rezultate dintr-o asamblare sau linkeditare, dar şi în alte scopuri. Menţionăm cîteva utilizări ale acestor programe:

— vizualizarea conţinutului unor adrese sistem sau unor secvenţe din nucleul sistem;

— încărcarea în memorie şi vizualizarea conţinutului unor fişiere de orice tip, inclusiv fişiere sursă, fişiere text, fişiere obiect relocabile, fişiere de date etc.;

— modificarea unor programe în format executabil, pentru care nu există forma sursă sau pentru a evita editarea şi reasamblarea unor programe mari;

— dezasamblarea unor programe existente numai în format executabil;

— introducerea şi testarea unor secvenţe scurte de instrucţiuni (cu coduri simbolice), mult mai rapid decît prin procedura clasică de editare, asamblare, depanare.

Ca mod de lucru, precizăm că după ce se încarcă la adresa 100H (ca orice program dintr-un fişier COM), programele DDT şi SID se mută la sfîrşitul zonei TPA (cu modificarea corespunzătoare a adresei din octeţii 5 şi 6), lăsînd liberă zona TPA pentru încărcarea altor programe.

Apelarea programelor de depanare se poate face în două feluri:

— prin comanda: DDT sau SID

— prin comanda: DDT nume sau SID numeh numes

unde „nume" este numele fişierului încărcat automat de către DDT sau SID (poate include discul suport şi trebuie să includă extensia numelui), „numeh" şi „numes" sînt numele fişierelor de tip „HEX" şi respectiv de tip „SYM" citite de către SID.

Programul SID poate fi folosit şi fără a i se transmite numele unui fişier tabelă de simboli, dar atunci nu se mai pot utiliza nume simbolice în comenzile SID.

După încărcarea fişierului din comandă se afişează la consolă prima adresă liberă (NEXT) şi adresa de lansare (PC), după care apare caracterul „—" („#" la SID) cu rolul de a indica utilizatorului că depanatorul este gata să primească o comandă.

Este posibilă şi încărcarea unui fişier prin comenzi DDT, după cum se pot încărca succesiv mai multe fişiere la adrese diferite sau la o aceeaşi adresă. Exemple:

1) DDT
   — ITEST.HEX
   — R

este absolut echivalentă cu DDT TEST.HEX

2) DDT CPM64.COM
   — IBIOSTPD.HEX
   — R4000

este o secvenţă folosită la generarea unui nou sistem CP/M şi încarcă un nou BIOS (din fişierul BIOSTPD.HEX) peste o parte din nucleul CP/M (adus din fişierul CPM64.COM).

5.4.2. Comenzi DDT şi SID

O comandă DDT (SID) poate fi urmată de unul sau mai mulţi parametri separaţi prin virgule sau spaţii; numărul de parametri depinde de tipul comenzii şi de scopul în care este folosită o comandă. De exemplu, comanda X poate să fie sau nu urmată de un nume de registru ca parametru.

Toţi parametrii numerici (adrese de memorie sau date de un octet) se introduc în hexa şi afişările de valori numerice de către DDT se fac tot în hexazecimal.

Fiecare comandă este terminată cu caracterul 〈CR〉, iar pe parcursul introducerii comenzilor se pot utiliza caracterele de control următoare:

BS = şterge ultimul caracter introdus;
CTRL/U, CTRL/X = şterge ultima linie introdusă;
CTRL/C = reiniţializare sistem;

Ieşirea din DDT se poate face prin CTRL/C sau prin comanda G0 (salt la adresa 0); imediat după ieşire se poate folosi comanda SAVE pentru salvarea programului depanat pe disc.

A = *Asamblare* (Assembly)

Comanda „A" are forma generală:

— As

unde „s" este adresa de implantare a codului simbolic ce urmează a fi introdus de operator în continuare.

DDT afişează apoi adresa şi aşteaptă codul mnemonic 8080 urmat de operanzi sub formă numerică (hexazecimală) şi (sau) nume de registre. În felul acesta se pot introduce mai multe linii de cod, pînă cînd se răspunde la adresă cu caracterul 〈CR〉 (o linie vidă). Verificarea codului introdus se poate face prin comanda L.

D = *Afişare* conţinut memorie (Display memory)

Comanda „D" poate avea una dintre formele:

— D
— Ds
— Ds, f

Forma „D" afişează 16 linii (a cîte 16 octeţi fiecare) de la adresa curentă menţinută de DDT (iniţial această adresă este 100H).

Forma „Ds" afişează 16 linii începînd cu adresa de start „s".

Forma „Ds, f" afişează conţinutul memoriei cuprinse între adresele „s" şi „f".

Utilizarea repetată a comenzii „D" duce la afişarea de zone succesive de memorie, deoarece adresa curentă de afişare rămîne la valoarea următoare ultimei adrese vizualizate.

Afişarea poate fi terminată forţat prin apăsarea tastei RUB (DEL) sau poate fi suspendată temporar şi apoi reluată cu CTRL/S.

F = *Umplere* zonă memorie (Fill memory)

Comanda „F" are forma următoare:

— Fs, f, c

Efectul acestei comenzi este de a introduce valoarea „c" în memorie între adresele „s" şi „f", inclusiv.

G = *Execuţie* instrucţiuni (Go)

Comanda „G" poate avea una dintre următoarele forme:

— G
— Gs
— Gs, b
— Gs, b, c
— G, b
— G, b, c

unde „s" este adresa de unde se începe executarea unei secvenţe de instrucţiuni. iar „b" şi „c" sînt adrese de oprire a execuţiei („Break points").

Dacă nu se specifică adresa „s" atunci execuţia începe de la adresa curentă (din registrul PC) pentru forma „G" sau continuă de la ultimul punct de oprire pentru formele „G, b, c" şi „G, b".

Sînt prevăzute două adrese „b" şi „c" pentru testarea ramificărilor din program pentru care nu se ştie sigur pe care ramură continuă execuţia programului.

După oprirea într-un „breakpoint" se afişează adresa de oprire şi se pot utiliza orice comenzi DDT.

I = *Stabilire fişier* de intrare (Input file)

Comanda „I" are forma:

— I <nume.tip>

unde <nume.tip> este identificatorul unui fişier.

Efectul comenzii este de a introduce numele de fişier în FCB-ul implicit, de la adresa 5CH. Acest bloc de control poate fi utilizat de

către DDT la o comandă de citire (R) pentru încărcarea în memorie a unui nou program sau poate fi utilizat de către programul depanat.

L= *Listare* cod simbolic (List)

Comanda „L" poate avea una dintre formele următoare:

— L
— Ls
— Ls, f

Forma „L" listează 12 linii de cod mașină dezasamblat începînd de la adresa curentă de listare (inițial 100 H).

Forma „Ls" listează 12 instrucțiuni simbolice începînd de la adresa de start „s".

Forma „Ls, f" dezasamblează și listează sub formă de cod simbolic conținutul zonei de memorie cuprinse între adresa „s" și adresa „f".

Utilizarea repetată a comenzii „L" afișează zone succesive de memorie.

O listare mai lungă poate fi terminată forțat prin tastarea clapei DEL (RUB).

M = *Mutare* în memorie (Move)

Comanda „M" are forma:

— Ms, f, d

și mută conținutul zonei de memorie dintre adresele „s" și „f" în zona de memorie care începe cu adresa „d".

R = *Citire fișier* (Read file)

Comanda „R" poate avea două forme :

— R
— Rb

unde „b" este o bază adunată la adresa de încărcare (pentru forma „R" această adresă este 0). Efectul comenzii este de a citi în memorie un fișier de pe disc (de tip HEX sau COM), desemnat printr-o comandă „I" anterioară.

Adresa de încărcare se află pe disc pentru fișierele de tip „HEX" și este implicit 100H pentru fișierele de tip „COM".

Fișierul citit conține de obicei programul depanat sau o parte din acest program.

S = *Înlocuiește* în memorie (Substitute)

Comanda „S" are forma:

— Sa

unde „a" este o adresă de memorie (în hexazecimal). Ca răspuns DDT afișează adresa de memorie și conținutul ei apoi așteaptă noul conținut; dacă nu dorește modificarea conținutului afișat, operatorul poate introduce caracterul ⟨CR⟩. În ambele cazuri programul DDT va afișa pe o linie nouă adresa următoare și conținutul acesteia repetînd acest lucru pînă cînd operatorul introduce caracterul „." sau alt caracter incorect.

T = *Trasare* (Trace)
Comanda „T" poate avea două forme:

— T
— Tn

Prima formă permite execuția pas cu pas a unui program; efectul comenzii „T" fiind executarea instrucțiunii de la adresa curentă și afișarea adresei imediat următoare (adresa de oprire).

Valoarea din registrele HL este luată de către DDT ca adresă de afișare pentru comanda „D", iar adresa de oprire este luată ca adresă de listare pentru comanda „L".

Forma „Tn" execută „n" instrucțiuni și apoi oprește programul într-un breakpoint. Se mai poate opri execuția programului lansat prin „Tn" apăsînd clapa DEL care forțează un breakpoint (de exemplu, într-un ciclu fără sfîrșit).

U = *Trasare* fără afișare (Untrace)
Comanda „U" este similară cu comanda „T", de care diferă prin aceea că nu se afișează pașii de program intermediari.

X = *afișează registre* (Examine)
Comanda „X" poate avea următoarele forme:

— X
— Xr

Forma „X" afișează conținutul registrelor lui 8080 și valorile indicatorilor de condiții.

Forma „Xr" afișează și permite modificarea unui registru „r" sau unui indicator de condiție.

Parametrul „r" poate avea următoarele valori:
A, B, D, H, P, S pentru registrele A BC, DE, HL, PC, SP.
C, Z, M, E, I pentru indicatorii CY, Z, S, P și CY auxiliar (transport între cifrele binar zecimale ale unui octet).

5.4.3. Diferențe între programele SID, ZSID și DDT

— Caracterul prompt pentru (Z)SID este „#" și nu „—".

— În comanda de lansare a programelor SID și ZSID pot apărea unul, două sau trei nume de fișiere, separate prin blancuri:

SID fhex fsym futl

unde:

„fhex" este fișierul de tip „HEX" sau „COM" ce conține programul care se depanează;

„fsym" este fișierul de tip „SYM" obținut la asamblarea programului depanat (cu ASM sau cu MAC) și care conține tabela de simboli pentru acest program;

„futl" este un fișier de tip „UTL" care conține subrutine utilitare necesare în depanare.

— Dacă s-a folosit un fișier de tip „SYM" în comanda (Z)SID, atunci programul (Z)SID va utiliza num simbolice din acest fișier la

dezasamblarea programului din memorie (comanda L). De asemenea se pot utiliza nume simbolice în comenzile SID care folosesc ca parametri adrese de memorie: A, D G, F, M, S.

— Programul DDT acceptă numai constante exprimate în hexazecimal, programele SID și ZSID recunosc următoarele tipuri de constante:

— numere hexazecimale (41);

— numere zecimale precedate de caracterul „#" (#65);

— caractere ASCII încadrate de ghilimele simple ('A');

— nume simbolice precedate de caracterul „." (.LOOP).

Aceste constante pot fi utilizate atît în comenzile (Z)SID cît și în programele simbolice introduse după comanda A (asamblare).

— Programele SID, ZSID recunosc 3 noi comenzi:

Ca (CALL) apel de subrutină de la adresa „a";

Hc (HEX) afișează echivalentul în hexazecimal, zecimal și ASCII al constantei „c";

Hc1 + c2 sau Hc1—c2 calculează suma sau diferența a două constante în hexazecimal;

Pa n (PASS) inserează un punct de oprire („breakpoint") după „n" treceri prin adresa „a" (adresa simbolică sau numerică). La fiecare trecere prin adresa respectivă se afișează „PASS n" și starea registrelor apoi se decrementează contorul „n".

— Comanda I (Input), la SID, completează la adresa 80H numărul de caractere introduse după litera I, iar la adresa 81H și următoarele pune aceste caractere (coada comenzii).

6. PROGRAMARE ÎN LIMBAJ MAŞINĂ 8080 ŞI Z80

Programarea în limbaj maşină prezintă multe aspecte specifice faţă de programarea în limbajele evoluate, universale, iar tehnicile de programare folosite sînt în mare măsură determinate de particularităţile procesorului: set de instrucţiuni, moduri de adresare ş.a.

Utilizarea instrucţiunilor de transfer, de comparaţie, aritmetice, logice, modurile de adresare a operanzilor din memorie, programarea ciclurilor, utilizarea stivei, transmiterea de date la subrutine, utilizarea directivelor de asamblare ş.a. pot fi considerate ca tehnici de programare de bază, aplicabile la orice echipament cu microprocesor 8080 sau Z80 şi independente de suportul software folosit.

Ca tehnici de programare avansate în limbaj de programare putem considera: definirea şi utilizarea de macroinstrucţiuni, programarea modulară, utilizarea directivelor de asamblare mai speciale, programarea operaţiilor de prelucrare a fişierelor secvenţiale, prelucrarea unor structuri de date mai complexe ş.a.

Pe lîngă aceste tehnici de programare generale, anumite clase de aplicaţii necesită cunoaşterea şi folosirea unor tehnici de programare specifice acestor aplicaţii, ca de exemplu: operaţii de prelucrare a unor fişiere cu articole de lungime oarecare, tehnici de acces selectiv la articole pe bază de cheie, calcule cu numere neîntregi reprezentate în binar virgulă mobilă, calcule cu numere întregi mai lungi de doi octeţi, calcule cu numere zecimale reprezentate în binar zecimal (BCD) sau în ASCII, programare concurentă cu taskuri paralele, operaţii grafice de generare sau de analiză a imaginilor etc.

Programarea operaţiilor de intrare-ieşire este discutată în capitolul următor, în contextul sistemului de operare CP/M.

Descrierea completă a setului de instrucţiuni 8080 şi, respectiv, Z80 se poate găsi în mai multe manuale, foi de catalog sau ghiduri de utilizare referitoare la aceste microprocesoare şi, ca urmare, în acest capitol se comentează şi se exemplifică utilizarea instrucţiunilor maşină în diferite situaţii practice de programare.

În mod deliberat s-au folosit ca exemple numai secvențe de instrucțiuni sau subrutine scurte și ușor de urmărit, dar desprinse din programe reale, fără ca prin aceasta să se nege utilitatea citirii unor programe complete, mai mari, care să rezolve integral o problemă dată.

6.1. Utilizarea instrucțiunilor mașină 8080

6.1.1. Scurtă descriere a microprocesorului 8080

Procesorul 8080 are 7 registre cu lungime de 8 biți, accesibile programatorului prin numele A, B, C, D, E, H, L.

Registrul A este registrul acumulator pentru operațiile pe 8 biți, în sensul că primul operand și rezultatul operației folosesc obligatoriu registrul A.

Celelalte 6 registre pot fi utilizate individual sau, în anumite instrucțiuni, ca perechi de registre BC, DE, HL, avînd funcția unor registre cu lungimea de 16 biți.

Registrele HL sînt folosite de obicei pentru adresarea indirectă implicită a operanzilor din memorie, în sensul că majoritatea instrucțiunilor cu referire la memorie consideră implicit operandul la adresa aflată în registrele H și L. În aceste cazuri registrul H conține partea superioară a adresei („High"), iar registrul L conține partea inferioară din adresă („Low").

Procesorul 8080 are și două registre de 16 biți utilizabile direct sau implicit de anumite instrucțiuni: SP („Stack Pointer") și PC („Program Counter").

Registrul PC conține adresa instrucțiunii care urmează să se execute, iar registrul SP se folosește pentru adresarea unei zone de memorie organizate ca stivă de cuvinte: adresa din SP este adresa cuvîntului din vîrful stivei.

Memoria internă este organizată pe octeți, deci adresele datelor sau instrucțiunilor din memorie sînt adrese de octet.

Un cuvînt este o pereche de doi octeți succesivi; un cuvînt de date poate conține o adresă de memorie, un număr întreg de 16 biți sau două caractere.

Spre deosebire de alte procesoare, la 8080 un cuvînt se poate memora la orice adresă (pară sau impară), iar instrucțiunile cu operanzi de un cuvînt se pot referi la orice adresă.

Memorarea celor doi octeți dintr-un cuvînt se face de obicei cu octetul inferior la adresa mai mică și cu octetul superior la adresa imediat următoare; această ordine de memorare se întîlnește atît la datele memorate cu instrucțiuni pe un cuvînt (SHLD), cît și în cadrul instrucțiunilor

care conțin o adresă de memorie. La transferul unui cuvînt între memorie și perechea de registre HL (prin instrucțiuni LHLD, SHLD) se face automat inversarea celor doi octeți din cuvînt, astfel încît octetul inferior se schimbă cu L, iar octetul superior cu H.

Deși este posibil ca prin instrucțiuni de transfer pe un octet să se memoreze octeții unui cuvînt în orice ordine, este bine să se respecte convenția de memorare inferior-superior, pentru uniformitate și pentru evitarea unor erori.

Instrucțiunile procesorului 8080 pot fi clasificate după lungimea lor în trei categorii:

— instrucțiuni de un octet, fără operand sau cu operand într-un registru;

— instrucțiuni de doi octeți, cu operand imediat de un octet care se memorează în al doilea octet din instrucțiune;

— instrucțiuni de trei octeți, cu operand imediat de doi octeți sau cu referire directă la memorie; valoarea operandului sau adresei se memorează în octeții doi și trei din instrucțiune, cu partea inferioară în octetul doi.

O serie de instrucțiuni de un octet sînt instrucțiuni cu referire la memorie, care conțin doar o indicație asupra modului de adresare (prin litera M în format extern); adresa operandului este considerată implicit în registrele H și L.

Ca o particularitate a instrucțiunilor 8080 cu doi operanzi menționăm ordinea celor doi operanzi în forma internă și în forma externă a instrucțiunilor: mai întîi operandul destinație și apoi operandul sursă.

La microprocesoarele pe 8 biți marea majoritate a instrucțiunilor operează cu operanzi de un octet, ceea ce este foarte util în prelucrarea șirurilor de caractere (a textelor).

Instrucțiunile pe cuvinte (de 16 biți) au fost introduse în principal pentru manipularea unor adrese de memorie, dar pot fi folosite și pentru numere întregi pe 16 biți.

Toate operațiile de intrare-ieșire se realizează prin intermediul a 256 porturi de I/E și a două instrucțiuni care pot transfera un octet între registrul A și un port de I/E specificat.

După executarea unor instrucțiuni de prelucrare (aritmetice, logice, de comparație și deplasare) se poziționează o parte sau toți *indicatorii de condiții*, în funcție de rezultatul operației respective. Există însă și instrucțiuni care nu modifică starea indicatorilor de condiții, cum ar fi de exemplu instrucțiunile de transfer al datelor între memorie și registre sau între registre.

Deși poziționarea indicatorilor de condiții este specifică fiecărei instrucțiuni, se poate indica următoarea semnificație generală a acestor indicatori:

CY (carry)
indică un transport de la bitul cel mai semnificativ, ceea ce se întîmplă şi la scădere sau comparaţie, cînd primul operand (din A) este mai mic decît al doilea operand.

Z (zero)
indică un rezultat nul prin valoarea Z = 1 şi un rezultat nenul prin valoarea Z = 0 (rezultatul este de obicei în A).

S (sign)
indică semnul rezultatului, care este şi bitul cel mai semnificativ din A.

P (parity)
indică paritatea numărului de biţi egali cu 1 din registrul A : P = 1 dacă există un număr par de biţi 1 în registrul A.

Starea indicatorilor de condiţii poate fi testată prin instrucţiunile de salt condiţionat, de apel condiţionat de subrutină şi de revenire condiţionată din subrutină.

Registrul A, ca octet superior, şi indicatorii de condiţii, ca octet inferior, formează *cuvîntul de stare* program PSW.

6.1.2. Instrucţiunile microprocesorului 8080

Instrucţiunile maşină pot fi grupate după funcţiile lor în trei categorii importante:

— instrucţiuni de mutare şi de prelucrare a datelor;
— instrucţiuni de control (de salt);
— instrucţiuni de intrare-ieşire.

Cele mai utilizate instrucţiuni maşină sînt instrucţiunile de transfer a datelor şi instrucţiunile de salt.

Instrucţiunile de transfer, de tipul „Move" sau „Load" asigură copierea de octeţi sau de cuvinte între registre sau între registre şi memorie.

Iniţial toate datele programului se află în memorie; tot în memorie se trimit rezultatele finale şi unele rezultate intermediare ale prelucrărilor Pe de altă parte, toate instrucţiunile de prelucrare 8080 şi Z80 cer ca unul dintre operanzi să fie într-un registru sau într-o pereche de registre; de aici rezultă şi utilitatea instrucţiunilor de încărcare (memorare) care pregătesc instrucţiunile de prelucrare şi apoi salvează în memorie rezultatele prelucrărilor.

Instrucţiunile de transfer pe octeţi nu modifică starea indicatorilor de condiţii şi se folosesc pentru:

— copierea conţinutului unui registru într-un alt registru sau în memorie (pentru salvarea registrului);
— încărcarea într-un registru (de obicei A) a unui octet din memorie sau din alt registru în vederea prelucrării sau comparării lui;
— încărcarea într-un registru sau în memorie a unei valori constante, care urmează să fie folosită într-o operaţie.

Instrucţiunile de încărcare şi de memorare a registrelor care folosesc adresarea indirectă prin HL tratează la fel toate registrele procesorului:

MOV r2, r1 ;mută în r2 din r1
MOV r, M ;mută în r de la adresa din HL
MOV M, r ;mută la adresa din HL din r
MVI r, n ;încarcă în r valoarea n de un octet
MVI M, n ;memorează la adresa din HL valoarea n

În instrucţiunile de mai sus „r" , „r1" şi „r2" pot fi orice nume de registru de 8 biţi.

Instrucţiunile de încărcare şi memorare directă sau indirectă prin BC şi DE pot lucra doar cu registrul A, ceea ce se reflectă şi în codul lor mnemonic, diferit de codul mnemonic al instrucţiunilor de transfer cu adresare prin HL:

LDA adr ;încarcă în A de la adresa adr
STA adr ;memorează din A la adresa adr
LDAX B ;încarcă în A de la adresa din BC
LDAX D ;încarcă în A de la adresa din DE
STAX B ;memorează pe A la adresa din BC
STAX D ;memorează pe A la adresa din DE

Instrucţiunile de încărcare (memorare) pe cuvinte sînt mai limitate ca posibilităţi în raport cu instrucţiunile pe octeţi şi pot fi clasificate în următoarele categorii:

— Instrucţiuni de încărcare şi memorare a registrelor HL cu adredresare directă a memoriei:

LHLD adr ;încarcă în HL un cuvînt de la adresa adr
SHLD adr ;memorează din HL un cuvînt la adresa adr

— Instrucţiuni de încărcare a unei constante de 16 biţi într-o pereche de registre sau în SP:

LXI rp, nn ;încarcă în perechea de registre rp valoarea nn
LXI SP, nn ;încarcă în SP valoarea nn

— Instrucţiuni de încărcare şi memorare a unei perechi de registre de la (la) adresa conţinută în SP cu incrementarea şi respectiv decrementarea registrului SP, care sînt instrucţiuni de lucru cu o stivă de cuvinte:

PUSH rp ;pune din rp în stiva adresată de SP
PUSH PSW ;pune A şi indicatorii în stiva adresată de SP
POP rp ;încarcă în rp cuvîntul din vîrful stivei
POP PSW ;încarcă în A şi în indicatori, din stivă

— Instrucţiuni de transfer şi de interschimb între perechi de registre sau registre de 16 biţi:

XCHG ;interschimb între HL şi DE
PCHL ;transfer în PC din HL
SPHL ;transfer în SP din HL
XTHL ;interschimb între HL şi vîrful stivei

Cu „rp" s-a notat o pereche de registre BC, DE sau HL.

Încărcarea în HL, BC şi DE se foloseşte în principal pentru pregătirea unei adresări indirecte prin aceste registre, iar valoarea încărcată reprezintă de obicei o adresă de memorie.

Instrucţiunile de tip PUSH au efectul următor: se scade 2 din SP şi se depune la adresa conţinută în SP un cuvînt dintr-o pereche de registre. Se folosesc pentru salvarea temporară a conţinutului registrelor.

Instrucţiunile de tip POP au efectul următor: se aduce cuvîntul de la adresa conţinută în SP într-o pereche de registre şi apoi se adună 2 la adresa din SP. Se folosesc pentru refacerea conţinutului anterior al unor registre salvate prin PUSH.

Zona de memorie adresată prin intermediul registrului SP se numeşte stivă („stack"), deoarece introducerea se poate face numai în vîrful stivei, iar extragerea tot din vîrful stivei (registrul SP conţine adresa vîrfului stivei).

Instrucţiunea XCHG este utilă pentru salvarea şi refacerea registrelor de adresare şi de calcul HL.

Instrucţiunea SPHL se foloseşte în principal pentru refacerea registrului SP; salvarea registrului SP în HL se face prin adunare cu DAD SP.

Exemplu:

Comutarea de pe o stivă pe altă stivă la începutul unui program şi refacerea stivei iniţiale la sfîrşitul programului:

```
; la început:
        LXI H,0 ;pentru transfer din SP în HL
        DAD  SP ;HL = SP
        SHLD OLDSTK ;salvare SP anterior
        LXI  SP, NEWSTK; comutare pe altă stivă
        ...
; la sfîrşit:
        LHLD OLDSTK ;încarcă în HL vechiul SP
        SPHL SP = HL ;revenire pe stiva iniţială
```

Instrucţiunea PCHL este de fapt o instrucţiune de salt la adresa din HL (salt indirect), folosită de exemplu la selectarea unei adrese calculate într-un tabel de adrese.

Instrucţiunea XTHL este utilă de exemplu în preluarea argumentelor primite în stivă de către o subrutină.

Instrucţiunile de comparaţie, aritmetice şi logice au doi operanzi de cîte un octet, dintre care un operand trebuie să se afle în registrul A; rezultatul operaţiilor aritmetice şi logice rămîne de asemenea în registrul acumulator A şi se poziţionează toţi indicatorii de condiţii în funcţie de valoarea rezultatului.

Instrucţiunile de comparare a doi octeţi au trei forme:

```
CMP r  ;compară A cu registrul r
CMP M  ;compară A cu octetul de la adresa din HL
CPI n  ;compară A cu valoarea imediată n
```

Comparaţia este simultan aritmetică şi algebrică: indicatorul CY arată rezultatul comparaţiei aritmetice (operanzii interpretaţi ca numere fără semn), iar indicatorul S arată rezultatul comparaţiei algebrice (operanzi interpretaţi ca numere cu semn). Modul de interpretare a operanzilor şi deci indicatorul testat depinde de semnificaţia operanzilor în programul respectiv; de exemplu după compararea a două caractere sau a două adrese se va testa indicatorul CY. Exemplu:
Testarea unui caracter aflat în registrul A dacă este caracter afişabil sau caracter de control:

```
CPI ''      ;spaţiul este primul caracter afişabil
JC CTRL     ;CY = 1 dacă este caracter de control
...         ;CY = 0 dacă caracter afişabil
```

Numerele întregi cu semn au lungime de cel puţin doi octeţi şi de aceea tipul comparaţiei este important la compararea de cuvinte, care se programează prin instrucţiuni pe octeţi.

Numerele negative se reprezintă de obicei în cod complementar (faţă de 2), iar în această reprezentare un număr negativ apare mai mare ca valoare pe 16 biţi decît un număr pozitiv; în consecinţă indicatorul testat după comparaţie depinde de interpretarea cuvintelor comparate: CY după compararea de adrese (fără semn) şi S după compararea de numere algebrice. Exemplu: subrutina de comparare a două adrese date în HL şi BC cu poziţionarea indicatorilor Z şi CY:

```
CPHL: MOV A, H; compară octeţii superiori
      CMP D
      RNZ        ;ieşire cu CY = 1 dacă HL <DE
      MOV A, L;compară octeţii inferiori
      CMP E
      RET        ;ieşire cu CY = 1 dacă HL <DE şi Z = 1 dacă
                 ;HL = DE
```

Instrucţiunile aritmetice pe octeţi realizează doar operaţii de adunare şi scădere între registrul acumulator A şi un alt operand; ele modifică toţi indicatorii de condiţii.

```
ADD r   ;adună la A octetul din registrul r
ADD M   ;adună la A octetul de la adresa din HL
ADI n   ;adună la A octetul cu valoarea n
SUB r   ;scade din A octetul din registrul r
SUB M   ;scade din A octetul de la adresa din HL
SUI n   ;scade din A octetul cu valoarea n
```

Fiecare dintre aceste 6 instrucţiuni mai are cîte o variantă, la care în calcule intervine şi indicatorul CY, a cărui valoare se adună sau se scade la A:

```
ADC r  ;adună la A octetul din r şi CY
ADC M  ;adună la A octetul de la adresa din HL şi CY
ACI n  ;adună la A octetul n şi CY
```

SBB r ;scade din A octetul din r și CY
SBB M ;scade din A octetul de la adresa din HL și CY
SBI n ;scade din A octetul n și CY

Instrucțiunile de adunare și de scădere cu CY sînt introduse pentru programarea operațiilor aritmetice cu numere mai lungi de un octet, la care indicatorul CY memorează transportul sau împrumutul de la octeții de rang inferior către octeții de rang superior. Exemplu:
Subrutina de scădere a două numere întregi de cîte un cuvînt aflate în registrele HL (descăzut) și DE (scăzător) cu rezultat în HL:

```
SHLDE: MOV A, L ;scădere octeți inferiori
       SUB E
       MOV L,A
; CY conține acum un eventual împrumut de la rangul superior
       MOV  A, H ;scădere octeți superiori
       SBB  D
       MOV  H, A
       RET
```

Aceeași scădere pe 2 octeți se poate realiza și prin complementarea scăzătorului urmată de adunarea la descăzut cu instrucțiunea DAD; această ultimă variantă este preferabilă pentru scăderea unor constante. Exemplu:
Scăderea constantei 1000 din HL:

```
LXI B, —1000
DAD B
```

Reprezentarea numerelor negative în cod complementar (aici de către asamblor) este cea care permite reducerea scăderii la adunarea complementului.

Instrucțiunile de *incrementare* și *decrementare* realizează adunarea sau scăderea lui 1 la un operand de un octet sau de un cuvînt, folosind numai un octet față de doi octeți cît ar fi necesari pentru adunare sau scădere:

```
INR  r  ;adună 1 la octetul din registrul r
DCR  r  ;scade 1 din registrul r
INX  rp ;adună 1 la cuvîntul din perechea de registre rp
DCX  rp ;scade 1 din perechea de registre rp
INX  SP ;adună 1 la numărul din SP
DCX  SP ;scade 1 din registrul SP
```

O altă deosebire față de instrucțiunile aritmetice constă în modul de poziționare a indicatorilor: INR și DCR nu poziționează CY și P iar INX și DCX nu poziționează nici un indicator. Exemplu:

Un ciclu repetat de 1000 de ori cu contor în BC:

```
        LXI B, 1000 ;valoare contor de pași
CICLU:... ;instrucțiuni repetate ciclic
        DCX     B ;scade 1 din BC
```

```
MOV   A, B ;pentru testare dacă BC este zero
ORA     C
JNZ CICLU ;repetă dacă contor nenul
```

Instrucțiunea de adunare dublă DAD este singura operație aritmetică pe 16 biți, dar deosebit de utilă în calcule de adrese sau calcule cu numere algebrice. Folosește ca acumulator registrele HL, iar al doilea operand poate fi într-o altă pereche de registre sau în SP:

```
DAD rp ;adună la HL cuvîntul din rp
DAD SP ;adună la SP cuvîntul din SP
```

Adunare dublă DAD H poate fi folosită și pentru deplasare la stînga cu o poziție în HL.

Instrucțiunea de *ajustare zecimală* după adunare DAA a fost introdusă pentru a permite adunarea de numere în reprezentarea binar-zecimală (BCD); folosită după o instrucțiune de adunare pe octeți, DAA corectează rezultatul acestei adunări binare, astfel încît el să constituie rezultatul adunării a două cifre BCD cu transport în CY pentru rangul următor. Exemplu:
Incrementarea unui număr BCD de un octet aflat la adresa ORE:

```
LDA ORE    ;se aduce în A
ADI 1      ;nu se poate cu INR A
DAA ;corecție zecimală
STA ORE ;memorează ora incrementată
```

Instrucțiunea de complementare CMA a registrului A face inversarea bit cu bit a octetului din A, deci inversarea logică și nu negarea (produce complementul față de 255 al numărului binar din A).

Instrucțiunile logice realizează trei operații logice binare SI (produs logic), SAU (sumă logică), SAU-EXCLUSIV (sumă modulo 2) în cele 3 variante de adresare a operanzilor:

```
ANA r        ORA r        XRA r
ANA M        ORA M        XRA M
ANI n        ORI n        XRI n
```

Operația de produs logic („And") se folosește pentru extragerea selectivă a unor grupuri de biți dintr-un octet prin anularea selectivă a biților care nu interesează. Exemplu: Secvență de extragere a cvartetului superior din A:

```
MOV B,A ;salvare octet inițial
ANI 0F0H ;produs logic cu mască 11110000
```

Operația de sumă logică („Or") se folosește pentru punerea pe 1 a unor biți dintr-un octet și pentru reunirea a două configurații binare

într-un singur octet. Exemplu: Subrutină de transformare a unui număr binar de 4 biți într-un caracter ASCII, reprezentînd cifra hexa corespunzătoare:

```
BH1:     CPI  11   ;este o cifra zecimala ?
         JC   BH11 ;salt daca cifra zecimala
         SUI  9    ;
         ORI  40H  ;litera A are codul 41H
         RET
BH11:    ORI  30H  ;cifra 0 are codul 30H
         RET
```

Operația de sumă logică exclusivă („Exclusive Or") are două proprietăți utile: suma exclusivă a unui număr cu octet avînd toți biții 1 (FFH) dă inversul logic al acelui număr, suma exclusivă între numere identice are ca rezultat zero.

Toate instrucțiunile logice poziționează indicatorii Z și S corespunzător rezultatului operației și anulează indicatorul CY; acest efect conduce la utilizarea lor pentru ștergerea indicatorului CY.

Punerea pe 1 și complementarea indicatorului CY se face prin instrucțiuni speciale:

STC ;CY = 1
CMC ;Complementare CY

Instrucțiunile de deplasare la 8080 sînt numai deplasări circulare (în inel) cu o poziție, pe lungime de 8 biți (numai registrul A) sau pe 9 biți (registrul A prelungit cu indicatorul CY). Indicatorul CY primește bitul care iese prin deplasare, indiferent de sensul deplasării. Iată lista acestor instrucțiuni:

RLC ;rotație la stînga în A
RRC ;rotație la dreapta în A
RAL ;rotație la stînga A și CY
RAR ;rotație la dreapta A și CY

Deplasarea la stînga este echivalentă cu o înmulțire cu 2, iar deplasarea la dreapta echivalează cu o împărțire la 2.

Indicatorul CY poate fi folosit pentru introducerea de 1 sau de 0 la deplasare și pentru memorarea transportului de la un rang la altul în cazul deplasărilor pe cuvinte. Exemplu: Subrutină de deplasare deschisă la dreapta cu o poziție în HL:

```
RRHL:    MOV  A,H
         RRC       ;deplasare H
         MOV  H,A
         MOV  A,L
         RAR       ;deplasare L si CY
         MOV  L,A
         RET
```

Testarea sau poziţionarea selectivă de biţi dintr-un octet se poate face fie prin operaţii logice fie prin deplasări. Exemplu: Verificarea unui număr de un octet din A dacă este par sau nu se poate face prin aducerea bitului inferior în CY:

```
    RRC
    JC IMPAR
PAR:...
```

sau prin secvenţa:

```
    ANI 01H ;masca de selecţie pentru bitul 0
    JNZ IMPAR
PAR:...
```

Instrucţiunile de salt sînt instrucţiuni pe 3 octeţi şi deci trebuie evitate pe cît posibil. Instrucţiunea de salt necondiţionat este

JMP adr; salt la adresa adr

Instrucţiunile de salt condiţionat pot testa fiecare dintre indicatori sau complementul său, dar nu pot testa o combinaţie de indicatori:

JZ adr ;salt la zero (dacă Z = 1)

JNZ adr ;salt la diferit de zero (dacă Z = 0)

JC adr ;salt dacă CY = 1

JNC adr salt dacă CY = 0

JM adr ;salt la minus (dacă S = 1)

JP adr ;salt la plus (dacă S = 0)

JPE adr ;salt la paritate pară (dacă P = 1)

JPO adr ;salt la paritate impară (dacă P = 0)

De multe ori saltul la zero urmează după o comparaţie şi are sensul de ,,salt la egalitate'', iar saltul la CY = 1 are de obicei semnificaţia de ,,salt la mai mic''.

Instrucţiunile de apelare a unei subrutine şi de revenire din subrutină sînt tot instrucţiuni de salt cu următoarele particularităţi; o instrucţiune de apel memorează adresa imediat următoare (conţinutul registrului PC) în stiva adresată de SP şi apoi realizează saltul; o insstrucţiune de revenire din subrutină face salt la adresa din vîrful stivei.

Lista instrucţiunilor de apel şi revenire este destul de mare deoarece există şi instrucţiuni condiţionate de fiecare indicator:

```
CALL adr     RET; necondiţionat
CZ adr       RZ  ;  dacă Z = 1
CNZ adr      RNZ; dacă Z = 0
CC adr       RC; dacă CY = 1
CNC adr      RNC; dacă CY = 0
CM adr       RM; dacă S = 1
```

CP adr   RP; dacă S = 0
CPE adr   RPE; dacă P = 1
CPO adr   RPO; dacă P = 0

Motivul introducerii acestor apeluri şi reveniri condiţionate îl constituie reducerea lungimii programelor, prin reducerea numărului de salturi condiţionate, care sînt instrucţiuni lungi. Exemplu: Subrutină de transformare a unui număr binar de 4 biţi într-un caracter ASCII reprezentînd cifra hexa corespunzătoare:

```
BH1:    ADI  '0'      ;considera ca este o cifra zecimala
        CPI  '9'+1    ;a fost intr-adevar cifra zecimala?
        RC            ;iesire daca da
        ADI  7        ;corectie pentru cifrele A...F
        RET
```

Totuşi utilizarea lor este destul de redusă şi poate constitui chiar o sursă de erori; astfel, revenirea condiţionată din subrutină nu se poate face atunci cînd se salvează unul sau cîteva registre la intrarea în subrutină, deoarece este necesară refacerea stivei înainte de revenire.

6.1.3. Tehnici de programare specifice 8080

Caracteristic programării în limbaj maşină este faptul că pentru realizarea unor prelucrări sînt posibile mai multe soluţii dintre care unele sînt optime, dintr-un anumit punct de vedere.

Memoria şi viteza limitate ale microcalculatoarelor pe 8 biţi au făcut ca *lungimea programelor* şi *timpul lor de execuţie* să constituie principalele criterii de apreciere şi de optimizare; în consecinţă multe programe folosesc anumite observaţii asupra setului de instrucţiuni 80807, care conduc la reducerea lungimii programelor în cod maşină sau la reducerea timpului de rulare. Vom prezenta în continuare unele dintre aceste soluţii precum şi alte situaţii tipice din programele reale, un ele devenite clasice pentru programatorii în limbaj maşină.

Adresarea operanzilor de un octet din memorie se poate face în principiu fie prin adresare directă (LDA, STA), fie prin adresare indirectă (MOV, LDAX, STAX). Exemplu:
Incrementarea unei variabile de un octet din memorie:

LXI H,V; încarcă adresa variabilei V în HL
INR M; incrementare în memorie
sau
LDA V; încarcă valoarea variabilei V în A
INR A; incrementare în A
STA V; memorează din A la adresa V

Prima secvență are numai 4 octeți și necesită registrele HL, iar a doua secvență are 7 octeți și folosește doar registrul A.

La 8080 și Z80 este preferabilă în general adresarea indirectă prin HL față de adresarea directă, din punct de vedere al numărului de octeți de program; un octet pentru o instrucțiune MOV față de 3 octeți pentru o instrucțiune LDA sau STA. Se poate afirma chiar că adresarea indirectă prin HL reprezintă principala metodă de referire la date din memorie pentru aceste procesoare.

În prelucrarea unor structuri de date dinamice (cu pointeri) poate fi necesară o adresare dublu indirectă prin HL, ceea ce se poate realiza destul de eficient. Exemplu:
Încărcarea în A a unui octet de la adresa aflată în memorie la adresa din HL:

```
;incarca in HL adresa octetului dorit
        MOV  A,M        ;octetul inferior al adresei
        INX  H          ;adresa octetului superior
        MOV  H,M        ;octetul superior al adresei in H
        MOV  L,A        ;octetul inferior al adresei in L
;incarca in A valoarea octetului
        MOV  A,M
```

Prelucrarea unei liste de octeți (unui șir de octeți) se face de obicei cu adresare indirectă prin HL. Exemplu:
Subrutină de afișare a unui șir de caractere terminat cu un octet zero și a cărui adresă se dă în HL:

```
PUTS:   MOV  A,M        ;se aduce in A un caracter
        ORA  A          ;test daca sfirsit de sir
        RZ              ;terminare subrutina daca sfirsit de sir
        MOV  E,A        ;afisare dn reg.E
        CALL CONOUT     ;scrie caracter din E
        INX  H          ;adresa caracterului urmator
        JMP  PUTS
```

Prelucrarea în paralel a două sau trei zone de memorie se poate face cu adresare indirectă prin registre diferite sau prin salvarea și refacerea alternativă a registrelor HL în stivă. Exemplu:
Secvență de transfer a unui șir de caractere terminat cu zero de la adresa S1 la adresa S2:

```
        LXI  H,S1       ;adresa sursa in HL
        LXI  D,S2       ;adresa destinatie in DE
MVS:    MOV  A,M        ;incarca in A un caracter din sirul S1
        STAX D          ;memoreaza din A in sirul S2
        INX  H          ;adresa urmatoare in S1
        INX  D          ;adresa urmatoare in S2
        ORA  A          ;a fost un octet zero ?
        JNZ  MVS        ;repeta daca nu a fost zero
```

Aşa cum se observă în acest exemplu, dacă se prelucrează octeţi de la adrese succesive de memorie nu este necesară încărcarea repetată a registrelor de adresare, fiind suficientă incrementarea (sau decrementarea) adreselor din registre.

Deşi nu este prevăzută o adresare indexată la 8080, este posibilă o indexare prin adunarea unei adrese relative la o adresă de bază fixă. Dacă valoarea adresei relative este mai mică de 225, atunci calculul adresei absolute prin indexare se poate face cu subrutina următoare de adunare a lui A la HL. Exemplu:

```
ADHL:   ADD  L    ;aduna A cu L
        MOV  L,A  ;rezultatul adunarii trece in L
        RNC       ;gata daca nu exista transport
        INR  H    ;aduna transport de la L la H
        RET
```

Dacă valoarea adresei relative este mai mare ca 255, se va folosi instrucţiunea DAD pentru indexare.

Operanzii de un cuvînt din memorie pot fi adresaţi direct sau indirect prin HL. Exemplu:

Incrementarea unei variabile de un cuvînt din memorie:

```
LHLD   V  ;încarcă în HL de la adresa V
INX    H  ;incrementare în HL
SHLD   V  ;memorează HL la adresa V
```

Utilizarea de variabile în memorie poate fi redusă sau evitată uneori prin salvarea şi refacerea registrelor în stivă, ceea ce este preferabil avînd în vedere că instrucţiunile de lucru cu stiva (PUSH şi POP) sînt instrucţiuni scurte şi rapide. Exemplu: Contorul unui ciclu poate fi salvat şi readus în (din) stivă pentru a nu ocupa două registre pe toată durata ciclului:

```
        LXI   B,1000  ;initializare contor
CICLU:  PUSH  B       ;salvare contor
        ...           ;instructiuni din ciclu (pot folosi BC)
        POP   B       ;readucere contor in BC
        DCX   B       ;scade din BC
        MOV   A,B     ;test daca BC este zero
        ORA   C
        JNZ   CICLU   ;repeta daca BC nu este zero
```

Atunci cînd se foloseşte stiva pentru memorarea temporară a conţinutului unor registre, trebuie urmărită refacerea stării stivei; tot ce se pune în stivă trebuie scos şi asta pe toate ramurile posibile, în programele ramificate.

Specific pentru 8080 (şi Z80) este faptul că aproape toate instrucţiunile de transfer şi de prelucrare care pot lucra cu un registru (altul decît registrul A) pot lucra şi cu un octet din memorie într-un mod asemănător, instrucţiunile avînd aceeaşi lungime de un octet.

Această unificare a tratării operanzilor din memorie şi din registre se realizează prin adresarea indirectă, implicită prin registrele HL. Exemplu:
Căutarea unui caracter dat in reg. A în zona de la adresa SIR:

```
        LXI  H,SIR    ;adresa sir in HL
SKP:    MOV  B,M      ;se aduce un caracter din sir in B
        CMP  B        ;compara cu caracterul din a
        JZ   FOUND    ;salt daca s-a gasit
        INX  H        ;adresa caracterului urmator
        JMP  SKP      ;repeta cautarea
```

Aceeaşi căutare a caracterului din A în memorie se poate scrie mai scurt astfel:

```
        LXI  H,SIR    ;adresa sir in HL
SKP:    CMP  M        ;compara çaracterul din sir cu A ?
        JZ   FOUND    ;salt daca s-a gasit
        INX  H        ;adresa caracterului urmator
        JMP  SKP      ;repeta cautarea
```

Prezentăm în continuare cîteva *artificii de programare*, destinate reducerii numărului de instrucţiuni sau timpului de calcul în realizarea anumitor operaţii uzuale.

Introducerea constantei zero în registrul acumulator A se poate face explicit printr-o instrucţiune de doi octeţi:

MVI A,0

sau printr-o instrucţiune de un octet:

XRA A sau SUB A

Compararea conţinutului registrului A cu zero se poate face explicit prin instrucţiunea de doi octeţi:

CPI 0

sau prin instrucţiunea de un octet:

ORA A

care poziţionează indicatorii CY şi Z la fel ca şi instrucţiunea de comparaţie.

Compararea conţinutului unui registru oarecare cu zero se poate face printr-o secvenţă de forma

MOV A, B ;mută din B în A
ORA A

care distruge conţinutul anterior al registrului A sau prin secvenţa următoare, tot de doi octeţi, care nu afectează registrul A:

INR B

DCR B ; Z = 1 dacă B = 0

Anularea indicatorului CY se poate face prin orice instrucţiune logică:

ORA A

Această operaţie poate fi necesară înaintea unei rotaţii în A sau la ieşirea dintr-o subrutină.

Instrucţiunile de salt, de apel şi de revenire condiţionată testează un singur indicator de condiţie. Pentru a testa condiţii de tipul „mai mic sau egal" cu o constantă fără a folosi două salturi succesive (totalizînd 6 octeţi), se poate modifica cu 1 valoarea constantei cu care se compară, după care se foloseşte o singură instrucţiune de salt. Exemplu: Transformarea caracterului din A din litere mici în litere mari:
Prima soluţie foloseşte două salturi condiţionate succesive:

```
        CPI   'a'
        JC    MARE         ;salt daca nu e litera mica
        CPI   'z'
        JZ    MIC          ;salt daca este z
        JNC   MARE         ;salt daca nu e litera mica
MIC:    ANI   5FH
```

Soluţia următoare foloseşte artificiul menţionat:

```
CPI   'a'
JC    MARE       ;salt daca < a
CPI   'z'+1
JNC   MARE       ;salt daca > z
ANI   5FH        ;anuleaza bitul 5 daca este litera mica
```

O soluţie mai bună ca număr de octeţi constă în definirea unei subrutine care să treacă din litere mici în litere mari, pentru că se înlocuiesc instrucţiunile de salt de 3 octeţi cu instrucţiuni de revenire condiţionată de un octet:

```
TOUPPER:  CPI   'a'
          RC              ;CY=1 daca nu e litera mica
          CPI   'z'+1
          RNC             ;CY=0 daca nu e litera mica
          ANI   5FH       ;anuleaza bitul 5
          RET             ;iesire daca a fost litera mica
```

Operațiile de înmulțire și împărțire pot fi realizate prin subrutine generale sau, în anumite cazuri, prin secvențe mai rapide reducînd înmulțirea la deplasări și adunări. Exemplu:
Secvența de înmulțire cu 10 a numărului din registrul A:

```
RLC               ;*2
MOV   B,A         ;B=A*2
RLC               ;A*4
RLC               ;A*8
ADD   B           ;A*8+A*2=A*10
```

Exemplu: Secvență de înmulțire a unui cuvînt din HL cu 10:

```
MOV   D,H         ;DE=HL
MOV   E,L
DAD   H           ;HL * 2
DAD   H           ;HL * 4
DAD   D           ;HL * 5
DAD   H           ;HL * 10
```

Utilizarea de subrutine

O subrutină este o secvență de instrucțiuni, scrisă într-un loc și utilizată în alt loc din program, prin apelarea ei cu o instrucțiune CALL. Subrutina poate fi apelată o singură dată sau de mai multe ori. O subrutină este o unitate de program cu o funcție bine definită.

Definirea de subrutine conduce pe de o parte la programe modulare, mai ușor de citit și de modificat, iar pe de altă parte conduce la o economie de memorie, dacă subrutina este folosită de mai multe ori într-un program. Existența unei colecții de subrutine care să realizeze funcții uzuale comune mai multor programe permite reducerea efortului de programare și implementarea mai rapidă a unor noi programe.

În general o subrutină primește de la programul sau subprogramul care o apelează anumite date inițiale și înapoiază acestuia unul sau mai multe rezultate. Există mai multe moduri de comunicare a datelor între o subrutină și programul care o folosește, fiecare avînd avantajele și limitările sale:

a) *Comunicarea implicită*, prin date globale, comune are loc atunci cînd subrutina folosește direct numele (adresele) variabilelor din programul apelant.

Subrutină de copiere a unui şir de lungime dată de la o adresă sursă la o adresă destinaţie.

```
MOVE1:  LXI   H,SURSA       ;adresa sursa in HL
        LXI   D,DEST        ;adresa destinatie in DE
        MVI   B,LUNG        ;lungime sir in B
MV1:    MOV   A,M           ;un octet din zona sursa
        STAX  D             ;se transfera in zona destinatie
        INX   H             ;adresa urmatoare in sursa
        INX   D             ;adresa urmatoare in destinatie
        DCR   B             ;contor ciclu
        JNZ   MV1           ;repeta pina cind B=0
        RET
```

Apelarea acestei subrutine se face printr-o singură instrucţiune:
CALL MOVE1

Numele simbolice SURSA, DEST, LUNG sînt cunoscute ca semnificaţie atît de programul apelant, cît şi de subrutină, deci constituie date comune implicite.

Subrutina MOVE1, aşa cum a fost scrisă, nu poate fi folosită pentru a muta şi alte şiruri de octeţi, de la alte adrese, nici în alte programe şi nici chiar în acelaşi program.

Comunicarea explicită, prin argumente sau parametri, conduce la subrutine de utilizare mai generală, dar transmiterea şi preluarea parametrilor necesită o serie de instrucţiuni suplimentare, aşa cum se va arăta în continuare.

b) *Comunicarea prin registre* este cea mai eficientă metodă de transmitere a parametrilor, dar nu este posibilă decît atunci cînd sînt numai cîteva argumente, datorită numărului limitat de registre ale procesorului. Exemplu:

Subrutină de copiere a unui şir de caractere de lungime dată, care primeşte parametrii în registrele HL (sursă), DE (destinaţie) şi B (lungime).

```
MOVE2:  MOV   A,M
        STAX  D
        INX   H
        INX   D
        DCR   B
        JNZ   MOVE2
        RET
```

Apelarea acestei subrutine se va face printr-o secvenţă de transmitere a parametrilor de forma următoare:

```
LXI H, SIR1     ; adresă sursă
LXI D, SIR2     ; adresă destinaţie
MVI B, 80       ; lungime
CALL MOVE2      ; apel subrutină
```

În legătură cu utilizarea registrelor în subrutine, trebuie remarcat că multe erori de programare în limbaj mașină se datorează modificării involuntare a conținutului unor registre în subrutine, registre pe care programul apelant contează că rămîn neschimbate.

Ca urmare, se recomandă ca o tehnică generală de programare ca toate registrele folosite de o subrutină să fie salvate la început în stivă și să fie refăcute la ieșirea din subrutină.

Dacă este necesar ca registrele cu date inițiale ale subrutinei să rămînă nemodificate, atunci se vor salva și aceste registre în cadrul subrutinei. Exemplu:
Subrutină pentru determinarea lungimii unui șir terminat cu zero și aflat la adresa dată în HL; lungimea rămîne în reg. A

```
STRLEN: PUSH H      ;HL trebuie sa ramina neschimbat
        PUSH B      ;B va fi folosit in subrutina
        MVI  B,0    ;in B se calculeaza lungime sir
SL1:    MOV  A,M    ;un octet din sir in A
        ORA  A      ;este zero ?
        JZ   SL2    ;salt daca sfirsit sir
        INR  B      ;numara octeti nenuli
        INX  H      ;adresa urmatoare in sir
        JMP  SL1
SL2:    MOV  A,B    ;lungimea ramine in A
        POP  B      ;refacere BC dn programul apelant
        POP  H      ;refacere adresa sir
        RET
```

c) *Comunicarea printr-o zonă de memorie*, în care se pun adresele parametrilor reprezintă o soluție aplicabilă pentru orice număr de parametri, dar mai puțin eficientă. Adresa zonei de argumente se transmite printr-o pereche de registre (de exemplu HL).

Transmiterea de adrese și nu de valori pentru datele inițiale ale subrutinei are cîteva avantaje:

— parametrii au toți aceeași lungime (de 2 octeți);

— subrutina poate modifica date din programul apelant.

Exemplu: Subrutină de copiere a unui șir de octeți de lungime dată; la adresa primită prin HL se află în ordine: adresa sursă, adresa destinație și adresa lungimii șirului.

```
MOVE3:  CALL ARGUM  ;adresa sursa in DE
        PUSH D      ;adresa sursa in stiva
        CALL ARGUM  ;adresa destinatie in DE
        PUSH D      ;adresa destinatie in stiva
        CALL ARGUM  ;adresa lungime sir in DE
        LDAX D      ;lungime sir in A
        MOV  B,A    ;lungime sir in B
        POP  D      ;adresa destinatie in DE
        POP  H      ;adresa sursa in HL
MOVE2:                ;la fel ca in MOVE2
```

Subrutina de preluare a următorului argument în DE de la adresa din HL poate arăta astfel:

```
ARGUM:  MOV  E, M   ; octetul inferior al adresei
        INX  H
        MOV  D, M   ; octetul superior al adresei
        INX  H      ; adresa argumentului următor în HL
        RET
```

Secvența de apelare a subrutinei MOVE3 poate folosi instrucțiuni sau directive pentru depunerea argumentelor în memorie:

```
LXI     H, ARGLST
CALL    MOVE3
...
ARGLST: DW  SURSA, DEST, LUNG
LUNG:   DB  80
```

sau:

```
LXI  H,SURSA
SHLD ARGLST
LXI  H,DEST
SHLD ARGLST+2
LXI  H,LUNG
SHLD ARGLST+4
LXI  H,ARGLST
CALL MOVE3
```

În subrutina MOVES au fost preluate de la început toate argumentele și puse temporar în stivă, dar există și alte posibilități: memorarea parametrilor în variabile locale subrutinei, extragerea adreselor din zona de argumente pe măsură ce ele sînt necesare în subrutină ș.a.

Compilatoarele FORTRAN și BASIC folosesc o combinație a metodelor b) și c) pentru eficiență la subprograme cu maxim 3 argumente și generalitate la subprograme cu mai multe argumente.

d) *Comunicarea argumentelor prin stiva* adresată de registrul SP este o variantă a comunicării prin memorie, dar mai eficientă, datorită instrucțiunilor de lucru cu stiva.

Scoaterea argumentelor din stivă se poate face fie în subrutină, fie în programul apelant.

Exemplu: Subrutină de copiere a unui șir de caractere care primește în stivă două adrese — sursă, destinație — și lungimea.

```
MOVE4:  POP  H      ;adresa de revenire in HL
        POP  B      ;lungime sir in B
        POP  D      ;adresa destinatie in DE
        XTHL        ;adresa sursa in HL
MOVE2:              ;continua ca la MOVE2
```

Secvența de apelare a subrutinei MOVE4 arată astfel:

```
LXI  H.SURSA
PUSH H
LXI  H.DEST
PUSH H
MVI  B.80
PUSH B
CALL MOVE4
```

În cazul existenței a mai mult de 3 argumente, subrutina este mai complicată, deoarece trebuie să memoreze argumentele scoase din stivă în variabile locale.

Vom analiza acum și varianta în care scoaterea parametrilor din stivă se face în programul apelant, iar subrutina citește argumentele din stivă folosind o adresare indexată.

Exemplu: Subrutină de copiere a unui șir de caractere cu parametrii primiți în stivă:

```
MOVES:  LXI  H,2
        DAD  SP       ;HL=adresa arg  3
        MOV  C,M
        INX  H
        MOV  B,M      ;lungime sir in B
        INX  H
        MOV  E,M      ;adresa destinatie in DE
        INX  H
        MOV  D,M
        INX  H
        MOV  A,M      ;adresa sursa in HL
        INX  H
        MOV  H,M
        MOV  L,A
MOVE2:
```

De observat că argumentele ar putea fi citite din stivă în orice ordine, prin calculul fiecărei adrese în parte; de exemplu pentru adresa sursă:

LXI H, 6 ; adresă relativă argument 1
DAD SP

În această subrutină se pot insera secvențe de salvare și de refacere a registrelor (spre deosebire de varianta precedentă), dar atunci se modifică adresa relativă a ultimului argument față de conținutul lui SP.

Programul apelant trebuie să descarce stiva după apelarea subrutinei, fie prin instrucțiuni POP, fie prin readucerea registrului SP la valoarea dinaintea secvenței de apelare:

```
LXI   H, 6      ; 3 argumente a cîte 2 octeți
DAD   SP
SPHL            ; SP = SP + 6
```

Transmiterea de valori sau de adrese prin stivă este folosită de compilatoarele C și PASCAL.

În încheierea acestei discuții referitoare la utilizarea subrutinelor, rezumăm cîteva *observații* mai importante:

— Comunicarea implicită, prin referire directă la date globale din memorie sau din registre, este cea mai simplă și mai eficientă metodă de comunicare între o subrutină și programul care o folosește; subrutinele scrise în acest fel nu pot fi însă reutilizate fără modificări și în alte programe.

— Pentru evitarea interferențelor nedorite între subrutinele apelate și programul apelant există două căi:

— utilizarea de date în memorie, destul de incomodă și ineficientă la 8080;

— utilizarea intensă a registrelor cu salvarea și refacerea sistematică a registrelor în subrutine.

— Metodele generale de transmitere a parametrilor la subrutine prin memorie (inclusiv prin stivă) sînt mai puțin utilizate la programarea directă în limbaj mașină, datorită limitelor în adresarea datelor din memorie la 8080.

Această diferență în utilizarea subrutinelor între programele rezultate din compilări și programele scrise direct în cod mașină explică în parte și diferența mare de lungime între aceste două categorii de programe.

Mai general, se poate constata că programele scrise în limbaj mașină folosesc de multe ori soluții mai puțin generale, dar mai eficiente, comparativ cu programele generate automat din limbaje de nivel ridicat.

— Atunci cînd se apelează subrutine scrise în limbaj mașină din programe scrise în limbaje evoluate, trebuie cunoscute și respectate convențiile de comunicare a parametrilor, folosite de compilatoarele respective.

— Stiva implicită, adresată de registrul SP, se va folosi ca zonă de salvare a registrelor și a unor rezultate intermediare în subrutine, dar cu mare atenție, deoarece adresele de revenire din subrutine se salvează în aceeași stivă și pot rezulta erori greu de localizat la execuție.

— Indicatorii de condiții CY și Z pot fi utilizați pentru rezultatele unor subrutine sau pentru a recunoaște terminarea cu eroare a unor subrutine; de obicei ieșirea cu CY = 1 înseamnă terminare cu eroare, iar ieșirea cu CY = 0 terminare normală.

— Utilizarea de subrutine este legată uneori de comunicarea între module obiect, asamblate separat, sau chiar de comunicarea între segmente de program diferite.

6.2. Utilizarea instrucţiunilor specifice Z80

6.2.1. Scurtă descriere a microprocesorului Z80

Procesorul Zilog Z80 a fost conceput ca o dezvoltare şi o perfecţionare a procesorului Intel 8080, atît din punct de vedere al performanţelor, cît şi al facilităţilor oferite constructorilor de microsisteme şi programatorilor.

Z80 este perfect compatibil în jos cu 8080, în sensul că toate instrucţiunile 8080 se regăsesc cu acelaşi cod intern şi la Z80, ceea ce permite rularea pe Z80 a oricărui program scris iniţial pentru 8080. De altfel trebuie spus că majoritatea programelor existente pe 8080 (inclusiv nucleul CP/M şi multe programe de sistem CP/M) nu au mai fost rescrise, pentru a beneficia de avantajele oferite de Z80, fiind utilizate în aceeaşi formă pe ambele tipuri de microprocesoare.

Din punct de vedere al programării, contribuţiile cele mai importante aduse de Z80 sînt următoarele:

— un număr mare de instrucţiuni suplimentare;
— noi registre şi moduri de adresare a datelor din memorie;
— modificarea formei externe a instrucţiunilor: alte coduri mnemonice şi alte notaţii pentru operanzi, mai unitare şi mai raţionale decît la 8080.

Registrele suplimentare de reîmprospătare a memoriei dinamice (registrul R) şi de întreruperi (registrul I) sînt mai puţin interesante pentru programatori, dar prezintă interes faptul că toate registrele de 8 biţi au fost dublate prin introducerea unui nou set de registre secundare:

A′, B′, C′, D′, E′, H′, L′

Aceste registre sînt folosite în principal pentru salvarea şi refacerea registrelor principale, prin două instrucţiuni de interschimb a celor două seturi de registre, deoarece nu sînt prevăzute alte instrucţiuni de lucru cu registrele secundare.

Cea mai importantă inovaţie în structura lui Z80 pentru programatori este introducerea a două noi registre de adresare, de cîte 16 biţi, numite registre de indexare IX şi IY.

Registrele IX, IY permit adresarea indirectă sau adresarea indirectă şi indexată a datelor din memorie; ele pot fi folosite în toate instrucţiunile cu adresare indirectă prin HL de la 8080 şi în noile instrucţiuni cu adresare indirectă adăugate la Z80.

Prin *adresare indirectă indexată* se înţelege că adresa operandului se calculează din conţinutul registrului IX sau IY, la care se adaugă o valoare constantă (index) de un octet (cu semn), conţinută în instrucţiune.

Utilizarea registrelor IX, IY ca registre de adresare a impus introducerea de noi instrucţiuni, care permit efectuarea tuturor operaţiilor posibile înainte cu HL:

— încărcarea directă din memorie a unui cuvînt;
— memorarea directă la o adresă a conţinutului registrelor;
— încărcarea unei constante în registre;
— transfer între IX, IY şi SP;
— incrementarea şi decrementarea registrelor;
— adunarea pe 16 biţi cu rezultat în IX, IY;
— salvarea şi refacerea în stivă a registrelor;
— salt la conţinutul lui IX, IY (transfer în PC).

Alte instrucţiuni au fost introduse la Z80 pentru realizarea unor operaţii considerate uzuale sau necesare: instrucţiuni pe blocuri de memorie, instrucţiuni la nivel de bit, instrucţiuni de deplasare deschisă aritmetică şi logică, negare acumulator.

Instrucţiunile de salt scurte de la Z80 se bazează pe observaţia că majoritatea instrucţiunilor de salt condiţionat şi necondiţionat fac saltul peste un număr mic de instrucţiuni şi deci distanţa dintre instrucţiunea de salt şi destinaţia saltului (adresa relativă de salt) se poate memora pe un singur octet, în timp ce adresa absolută de salt ocupă doi octeţi. Instrucţiunile de salt relativ ocupă numai doi octeţi, dar sînt limitate la un salt de maximum 128 de octeţi înainte sau înapoi faţă de adresa instrucţiunii (mai exact, faţă de PC).

Tot o instrucţiune scurtă de salt este şi instrucţiunea care combină decrementarea unui registru şi saltul la rezultat diferit de zero, operaţii care apar mereu împreună la programarea ciclurilor cu contor de un octet.

Instrucţiunile pe blocuri de memorie permit copierea unui şir de octeţi şi compararea unui octet cu un şir de octeţi, prin incrementarea sau decrementarea automată a registrelor de adresare a operanzilor (HL şi DE la copiere, HL la comparare) şi prin decrementarea registrelor BC care conţin lungimea operanzilor; într-o variantă operaţia se opreşte automat atunci cînd contorul de octeţi din BC ajunge la zero, iar în altă variantă nu se ţine seama de conţinutul registrelor BC, revenind programatorului sarcina de oprire a ciclului implicit de copiere sau de comparare.

Ideea executării repetate a unei operaţii apare şi în unele instrucţiuni de intrare-ieşire nou apărute la Z80, care transferă o succesiune de octeţi de la sau la un port de I/E.

Instrucţiunile la nivel de bit permit testarea, ştergerea şi poziţionarea (setarea) selectivă a unui bit dintr-un octet folosind ca operand numărul bitului respectiv şi nu o mască de un octet ca în instrucţiunile logice.

Instrucțiunile pentru deplasare de la 8080 au fost completate cu noi tipuri de *deplasări deschise:* deplasare aritmetică, deplasare logică și deplasarea BCD, precum și cu posibilitatea ca octetul rotit sau deplasat să se afle în orice registru sau în memorie și nu numai în registrul A.

Adunarea dublă pe 16 biți are acum o variantă în care participă și indicatorul CY la adunare; s-a introdus de asemenea și o scădere cu CY pe 16 biți din HL.

Utilizarea modurilor proprii Z80 de tratare a întreruperilor a necesitat introducerea cîtorva instrucțiuni pentru stabilirea modului de întrerupere, pentru încărcarea registrului de întreruperi I și pentru revenirea din rutinele de tratare a întreruperilor mascabile și nemascabile.

6.2.2. Codurile simbolice de instrucțiuni Z80

Mărirea substanțială a repertoriului de instrucțiuni și apariția unor noi moduri de adresare a impus regîndirea formatului extern al instrucțiunilor. Cu această ocazie s-au modificat codurile mnemonice pentru marea majoritate a instrucțiunilor și, mai ales, s-a schimbat modul de specificare a adresării indirecte a operanzilor.

Codul mnemonic Zilog al unei instrucțiuni este determinat numai de funcția realizată, spre deosebire de codul Intel, în care intervine și modul de adresare, lungimea operandului, registrul folosit; în acest fel a fost posibilă reducerea numărului de coduri simbolice folosite și deci simplificarea codificării și citirii programelor în limbaj de asamblare. De exemplu, în locul mnemonicelor MOV, MVI, LDA, LHLD, LDAX, LXI, STA, SHLD, STAX se folosește acum codul LD pentru orice transfer de octeți sau de cuvinte și indiferent de modul de adresare.

Pentru instrucțiunile des utilizate s-au ales coduri scurte, de numai două litere: LD, CP, JP, JR.

Deși schimbarea oarecum nemotivată a unor mnemonice de la 8080 poate fi discutabilă, se poate afirma că formatul Zilog al instrucțiunilor este superior formatului Intel, fiind mai explicit în referirea operanzilor.

Formatul Zilog specifică întotdeauna explicit adresa sau valoarea operandului cu următoarele convenții:

— Adresarea indirectă prin HL folosește numele registrelor HL în locul literei M de la 8080; în general, un. nume de registru de 16 biți între paranteze indică o adresare indirectă prin registrul respectiv.

Exemple:

```
LD A, (HL)
LD (HL), A
```

— Numele unei perechi de registre are două litere, fiind format din numele ambelor registre. Exemple:

```
PUSH  BC
ADD   HL, DE
```

— Cuvîntul de stare format din registrul A și indicatori („Flags") este notat tot cu două litere AF în loc de PSW. Exemplu: PUSH AF

— O constantă numerică sau simbolică fără paranteze indică un operand imediat (se folosește valoarea operandului). Exemple:

```
LD A, 0FFH    ; echivalent cu MVI A, 0FFH
LD HL, LISTA  ; echivalent cu LXI H, LISTA
```

— O constantă numerică sau simbolică în paranteze, considerată pe 16 biți, indică adresa din memorie a operandului și se folosește în instrucțiunile cu adresare directă la memorie. Exemple:

```
LD A, (4)       ; echivalent cu LDA 4
LD HL, (N1)     ; echivalent cu LHLD N1
```

— Indicatorii de condiții și negațiile lor constituie primul operand al instrucțiunilor de salt, de apel și de revenire, care au cîte un singur cod mnemonic. Exemple:

```
JP    C, MIC    ; echivalent cu JC MIC
RET   Z         ; echivalent cu RZ
CALL  NZ, SUB   ; echivalent cu CNZ SUB
```

— Registrul A apare explicit ca operand, atunci cînd nceeași instrucțiune simbolică poate avea și alți operanzi, dar nu apare ca operand explicit, dacă este singura posibilitate. Exemple.

```
ADD   A, C      ; echivalent cu ADD C
CP    C         ; echivalent cu CMP C
OR    (HL)      ; echivalent cu ORA M
```

Dacă formatul Zilog nu este totuși atît de mult folosit cît ar merita și nu a reușit să înlocuiască formatul Intel pentru instrucțiunile comune, explicația constă în aceea că el a apărut prea tîrziu, cînd programatorii se obișnuiseră deja cu formatul Intel, pentru care existau asambloare și alte utilitare.

Pentru a facilita trecerea de la mnemonicele Intel la mnemonicele Zilog vom prezenta în paralel cele două forme pentru o serie de instrucțiuni și moduri de adresare uzuale:

INTEL	ZILOG
MOV A,M	LD A,(HL)
MOV M,A	LD (HL),A
MVI B,10	LD B,10
MVI M,0	LD (HL),0
LDA ADR	LD A,(ADR)
STA ADR	LD (ADR),A
LDAX B	LD A,(BC)
STAX B	LD (BC),A
LXI H,ADR	LD HL,ADR
LHLD ADR	LD HL,(ADR)
SHLD ADR	LD (ADR),HL
SPHL	LD SP,HL
XCHG	EX DE,HL
XTHL	EX (SP),HL
CMP M	CP (HL)
INR M	INC (HL)
INX H	INC HL
ADD B	ADD A,B
DAD D	ADD HL,DE
RAL	RL A
JMP ADR	JP ADR
JC ADR	JP C,ADR
JNZ ADR	JP NZ,ADR
PCHL	JP (HL)
CC ADR	CALL C,ADR
RNZ	RET NZ
STC	SCF
RST 7	RST 38H

Urmează prezentarea tuturor codurilor mnemonice Zilog, grupate pe funcții și pe lungimea operanzilor (cu „e" s-a notat valoarea index)

Instrucțiuni de încărcare pe 8 biți

LD r, s ; r = s ; s : r, n, (HL), (IX + e), (IY + e)
LD d, r ; d = r ; d : (HL), r, (IX + e), (IY + e)
LD d, n ; d = n ; d : (HL), (IX + e), (IY + e)
LD A, s ; A = s ; s : (BC), (DE), (nn), I, R
LD d, A ; d = A ; d : (BC), (DE), (nn), I, R

Instrucțiuni de încărcare pe 16 biți

LD dd, nn ; dd = nn ; dd : BC, DE, HL, SP, IX, IY
LD dd, (nn) ; dd = (nn); dd : BC, DE, HL, SP, IX IY,
LD (nn), ss ; (nn) = ss ; ss : BC, DE, HL, SP, IX, IY
LD SP, ss ; SP = ss ; ss : HL, IX, IY
PUSH ss ; pune ss în stivă; ss : BC, DE, HL, AF, IX, IY
POP dd ; scoate din stivă în dd ; dd : BC, DE, HL, AF, IX, IY

Instrucțiuni de interschimb

EX	DE, HL	; interschimb DE cu HL
EX	AF, AF′	; interschimb AF cu AF′
EXX		; interschimb BC cu BC′, DE cu DE′, HL cu HL′
EX	(SP), ss	; interschimb ss cu vîrful stivei ; ss : HL, IX, IY

Instrucțiuni de mutare pe blocuri

LDI		; (DE) = (HL), DE = DE + 1, HL = HL + 1, BC = BC − 1
LDIR		; (DE) = (HL), DE = DE + 1, HL = HL + 1, BC = BC − 1 pînă cînd BC = 0
LDD		; (DE) = (HL), DE = DE − 1, HL = HL − 1, BC = BC − 1
LDDR		; (DE) = (HL), DE = DE − 1, HL = HL − 1, BC = BC − 1 pînă cînd BC = 0

Instrucțiuni de căutare într-un bloc

CPI		; A − (HL), HL = HL + 1, BC = BC − 1
CPIR		; A − (HL), HL = HL + 1, BC = BC − 1 pînă cînd BC = 0 sau A = (HL)
CPD		; A − (HL), HL = HL − 1, BC = BC − 1
CPDR		; A − (HL), HL = HL − 1, BC = BC − 1 pînă cînd BC = 0 sau A = (HL)

Instrucțiuni aritmetice pe 8 biți

ADD	A, s	; A = A + s ; s : r, n, (HL), (IX + e), (IY + e)
ADC	A, s	; A = A + s + CY
SUB	s	; A = A − s
SBC	A, s	; A = A − s − CY
AND	s	; A = A si s
OR	s	; A = A sau s
XOR	s	; A = A sau-exclusiv s
CP	s	; A − s
INC	d	; d = d + 1 ; d : r, (HL), (IX + e), (IY + e)
DEC	d	; d = d − 1 ; d : r, (HL), (IX + e), (IY + e)

Instrucțiuni aritmetice pe 16 biți

ADD	HL, ss	; HL = HL + ss ; ss : BC, DE, HL, SP
ADC	HL, ss	; HL = HL + ss + CY
SBC	HL, ss	; HL = HL − ss − CY
ADD	IX, ss	; IX = IX + ss ; ss : BC, DE, IX, SP
ADD	IY, ss	; IY = IY + ss ; ss : BC, DE, IY, SP
INC	dd	; dd = dd + 1 ; dd : BC, DE, HL, SP, IX, IY
DEC	dd	; dd = dd − 1 ; dd : BC, DE, HL, SP, IX, IY

Instrucțiuni cu registrul A și indicatorii

```
DAA            ; conversie în BCD după adunare sau scădere
CPL            ; complementare în A
NEG            ; negare în A, A = 0 — A
CCF            ; complementare CY
SCF            ; pune CY pe 1
```

Instrucțiuni diverse

```
NOP            ; nici o operație
HALT           ; oprire procesor
DI             ; dezactivare întreruperi
EI             ; activare întreruperi
IM O           ; stabilire mod 0 de întrerupere
IM 1           ; stabilire mod 1 de întrerupere
IM 2           ; stabilire mod 2 de întrerupere
```

Instrucțiuni de rotație și deplasare

```
RLC   s        ; rotație stînga în s ; s : r, (HL), (IX + e), (IY + e)
RL    s        ; rotație stînga s și CY
RRC   s        ; rotație dreapta în s
RR    s        ; rotație dreapta s și CY
SLA   s        ; deplasare aritmetică stînga în s
SRA   s        ; deplasare aritmetică dreapta în s
SRL   s        ; deplasare logică dreapta în s
RLD            ; rotație stînga BCD în (HL) și A
RRD            ; rotație dreapta BCD în (HL) și A
```

Instrucțiuni la nivel de bit

```
BIT   b, s     ; test bit b din s ; s : r, (HL), (IX + e), (IY + e)
SET   b, s     ; pune pe 1 bitul b din s
RES   b, s     ; pune pe 0 bitul b din s
```

Instrucțiuni de salt

```
JP    nn       ; salt la nn, PC = nn
JP    cc, nn   ; salt la nn, dacă condiția cc adevărată
JR    e        ; salt la PC + e, PC = PC + e
JR    kk, e    ; salt la PC + e, dacă condiția kk adevărată
JP    (ss)     ; PC = ss; ss : HL, IX, IY
DJNZ  e        ; B = B — 1, dacă B = 0 continuă, altfel PC=PC
```

Condiția cc poate fi: NZ, Z, NC, C, PO, PE, P, M
Condiția kk poate fi: NZ, Z, NC, C

Instrucţiuni de apel subrutină

```
CALL  nn      ; salvare PC în stivă şi salt la nn
CALL  cc, nn  ; dacă condiţia cc e falsă, continuă, altfel CALL
   Condiţia cc poate fi: NZ, Z, NC, C, PO, PE, P, M
RST   nn      ; identic cu CALL, dar nn este cel mult 64
```

Instrucţiuni de revenire din subrutină

```
RET           ; salt la adresa din vîrful stivei
RET   cc      ; dacă condiţia cc e falsă continuă, altfel RET
RETI          ; revenire din întrerupere, la fel ca RET
RETN          ; revenire din întrerupere nemascabilă
```

Instrucţiuni de intrare-ieşire

```
IN    A, (n)  ; citire in A de la portul n
IN    r, (C)  ; citire in r de la portul cu număr dat în C
INI           ; (HL) = (C), HL = HL + 1, B = B - 1
INIR          ; (HL) = C, HL = HL - 1, B = B - 1, repetă pînă
                cînd B = 0
IND           ; (HL) = (C), HL = HL - 1, B = B - 1
INDR          ; (HL) = (C), HL = HL - 1, B = B - 1, repetă
                pînă cînd B = 0
OUT   (n), A  ; scrie din A la portul n
OUT   (C), r  ; scrie din r la portul cu număr dat în C
OUTI          ; (C) = (HL), HL = HL + 1, B = B - 1
OTIR          ; (C) = (HL), HL = HL + 1, B = B - 1, repetă
                pînă cînd B = 0
OUTD          ; (C) = (HL), HL = HL - 1, B = B - 1
OTDR          ; (C) = (HL), HL = HL - 1, B = B - 1, repetă
                pînă cînd B = 0
```

6.2.3. Tehnici de programare specifice Z80

Dacă rescrierea programelor 8080 pentru Z80 nu este în general justificată, în schimb la scrierea de noi programe destinate numai maşinilor cu Z80 se pot obţine performanţe mai bune prin utilizarea posibilităţilor specifice ale acestui microprocesor.

În general toate tehnicile de programare folosite la 8080 rămîn valabile şi pentru Z80, cu cîteva excepţii datorate noilor instrucţiuni: salvare şi refacere registre, operaţii pe blocuri de octeţi, cicluri cu contor, unele deplasări ş.a.

În anumite cazuri folosirea instrucţiunilor noi de la Z80 conduce la *programe mai explicite*, dar nu mai scurte decît cele scrise pentru

8080; un exemplu îl constituie folosirea instrucțiunilor la nivel de bit față de instrucțiunile logice, iar un alt exemplu îl constituie folosirea indexării în loc de incrementare și indirectare. Exemplu:

Secvență de aducere în HL a adresei aflate în memorie la adresa dată în HL.

La 8080:

```
MOV A, M ;octet inferior al adresei
INX H
MOV H, M ;octet superior al adresei în H
MOV L, A ;octet inferior în L
```

La Z80:

```
PUSH HL
POP IX            ; IX = HL
LD L, (IX + 0) ;octet inferior al adresei
LD H, (IX + 1) ;octet superior al adresei
```

Așa cum se observă în acest exemplu, transferul unui cuvînt între registrele IX, IY și o pereche de registre se poate face prin stivă cu instrucțiuni PUSH și POP.

Salvarea registrelor la intrarea în subrutine, precum și refacerea lor se poate face folosind setul de registre secundare în locul stivei, ceea ce înlocuiește 3 instrucțiuni PUSH sau POP printr-o singură instrucțiune EXX.

Subrutină de citire a unui octet de la consolă cu o funcție sistem (care modifică registrele).

```
CI:     EXX             ;salvare BC,DE,HL
        LD   C,1        ;cod functie BDOS
        CALL 5          ;citeste caracter in reg.A
        EXX             ;refacere registre salvate
        RET
```

Instrucțiunile pe blocuri de memorie își găsesc o largă utilizare în prelucrarea șirurilor de caractere (a textelor).

Șirurile de caractere au o lungime variabilă, iar delimitarea lor în memorie se face fie printr-un octet zero la sfîrșitul șirului (practicată în limbajul C), fie prin precedarea șirului de un octet sau de un cuvînt care să conțină lungimea șirului (ca în Pascal MT +).

Dacă se cunoaște lungimea șirurilor, atunci se vor utiliza instrucțiunile cu repetare automată pînă la lungimea zero: LDIR, LDDR, CPIR, CPDR. Exemplu:

Subrutină de copiere a unui şir de caractere de la adresa din HL la adresa din DE pe lungimea din BC:

```
CPYSTR: LDIR
        RET
```

Exemplu: Subrutină de comparare a două şiruri de caractere cu adresele date în HL şi DE şi lungime în BC.

```
CMPSTR: LD   A,(DE)     ;un octet din sirul 2
        CPI             ;compara cu un octet din sirul 1
        RET  NZ         ;iesire daca inegalitate
        LD   B          ;test daca BC=0
        CP   C
        RET  Z          ;z=1 la egalitate
        INC  DE         ;urmatorul octet din sirul 2
        JR   CMPSTR     ;repeta comparatia
```

Dacă se prelucrează şiruri terminate cu un octet zero sau cu alt terminator de şir, atunci se vor folosi instrucţiunile LDI, LDD, CPI, CPD. Exemplu:
Subrutină de copiere a unui şir terminat cu zero de la adresa din HL la adresa din DE, inclusiv terminatorul de şir.

```
STRCPY: LD   A,(HL)         ;un octet din sirul sursa
        OR   A              ;sfirsit de sir?
        RET  Z              ;iesire daca sfirsit sir
        LDI                 ;copiere octet dela (HL) la (DE)
        JR   STRCPY         ;repeta pentru octetul urmator
```

Exemplu: Subrutină de comparare a două şiruri terminate cu zero şi aflate la adresele din HL şi DE, cu rezultat în Z.

```
STRCMP: LD   A,(DE)
        INC  DE
        OR   A
        RET  Z          ;iesire cu Z=1 daca siruri egale
        CPI             ;compara un caracter din cele doua
        JR   Z,STRCMP   ;continua comparare daca sint egale
        RET             ;iesire cu Z=0 daca siruri diferite
```

Exemplu: Subrutina de concatenare a două şiruri terminate cu zero fiecare şi aflate la adresele din HL şi DE.

```
STRCAT: XOR  A          ;cauta sfirsit sir cu adresa in HL
STRC1:  CPI             ;cauta zero in sirul de la (HL)
        JR   NZ,STRC1   ;repeta cautare pina cind se gaseste
        DEC  HL         ;HL=adresa inainte de zero
        EX   DE,HL      ;DE=adresa destinatie, HL=adresa sursa
STRC2:  LDI             ;muta un caracter din (HL) in (DE)
        LD   A,(HL)     ;test sfirsit sir mutat
        OR   A          ;este zero in A ?
        JR   NZ,STRC2
        RET
```

Existența celor două *noi registre de adresare* IX, IY încurajează utilizarea de variabile în memorie și reduce necesitatea salvării — refacerii repetate a registrelor HL, de exemplu, atunci cînd se prelucrează în paralel mai multe zone de date. Exemplu:

Subrutină de extragere a unui subșir dintr-un șir; IX conține adresa șirului inițial, IY conține adresa unde se mută subșirul extras, B conține numărul de caractere extrase, DE conține poziția în șirul dat de unde se extrag caractere.

```
SUBSTR: ADD  IX,DE      ;daca DE contine pozitia relativa
SUBS1:  LD   A,(IX+0)   ;scoate un caracter din sir in A
        LD   (IY+0),A   ;muta caracter in noul sir
        INC  IX         ;adresa urmatoare in sirul dat
        INC  IY         ;adresa urmatoare in noul sir
        DJNZ SUBS1      ;repeta pina cind B=0
        RET
```

De reținut forma indirectării prin registrele IX, IY: se va scrie (IX + 0) și nu (IX).

Adresarea indexată cu registrele IX, IY are multiple utilizări pentru referirea la date situate la adrese succesive de memorie sau la o distanță fixă între ele, precum și pentru adresarea componentelor unor structuri de date.

Exemplu: La ordonarea unei liste de cuvinte care începe la adresa dată în IX prin metoda inversării elementelor vecine se pot folosi următoarele secvențe:

```
; aducerea în HL a cuvîntului de la adresa din IX
  LD  L, (IX + 0)
  LD  H, (IX + 1)
; aducerea în DE a cuvîntului următor
  LD  E, (IX + 2)
  LD  D, (IX+3)
; comparare HL cu DE
  ...
```

Exemplu: Subrutină de introducere a numărului de înregistrare pentru acces direct la un fișier în octeții 33, 34, 35 din blocul descriptor de fișier FCB.

```
; DE = adresa FCB, HL = număr înregistrare în fișier

RREC:   PUSH DE          ;IX=DE
        POP  IX
        LD   (IX+33),L
        LD   (IX+34),H
        LD   (IX+35),0
        RET
```

Exemplu: Secvenţă de adăugare a tipului COM la numele unui fişier aflat în blocul de control FCB:

```
LD    IX,FCB          ;adresa FCB in IX
LD    (IX+9),'C'      ;tip fisier in octetii 9,10,11
LD    (IX+10),'O'
LD    (IX+11),'M'
```

Exemplu: Subrutină de incrementare a unui număr de două cifre reprezentate în ASCII şi aflat la adresa dată în IY.

```
INCASC: LD    A,(IY+1)        ;test cifra unitati
        CP    '9'             ; daca este 9
        JR    Z,INCA1         ;salt daca 9 la unitati
        INC   (IY+1)          ;aduna 1 la cifra unitatilor
        RET                   ;iesire cu Z=0
INCA1:  LD    (IY+1),'0'      ;dupa 9 urmeaza 0
        LD    A,(IY+0)        ;cifra zecilor
        CP    '9'             ;test daca 9
        JR    Z,INCA2         ;salt daca 99
        INC   (IY+0)          ;aduna 1 la cifra zeci
        RET                   ;iesire cu Z=0
INCA2:  LD    (IY+0),'0'      ;100 dupa 99
        RET                   ;iesire cu Z=1 daca depasire
```

În ceea ce priveşte instrucţiunile de control este recomandată utilizarea de cîte ori este posibil a *instrucţiunilor scurte de salt*. De altfel, dacă sînt necesare instrucţiuni de salt lungi, atunci este un semn că subrutinele din componenţa programului au prea multe instrucţiuni şi deci sînt mai greu de stăpînit şi de modificat. Distanţa dintre o subrutină şi locul unde este apelată poate fi oricît de mare, deoarece instrucţiunile de apel sînt instrucţiuni de salt lungi, cu adresă absolută.

La programarea ciclurilor cu contor de un octet se va utiliza instrucţiunea DJNZ de 2 octeţi, în locul decrementării şi saltului lung, care însumează 4 octeţi la 8080. Exemplu:

Subrutină de afişare la consolă, în hexazecimal, a conţinutului unei zone de memorie cu cîte 16 octeţi pe linie.

```
HL=adresa initiala , DE=adresa finala
DMP:    EX    DE,HL     ;HL=adresa finala
        OR    A         ;CY=0 pentru scaderea urmatoare
        SBC   HL,DE     ;numar de octeti in HL
        EX    DE,HL     ;HL=adresa initiala, DE numar de octeti
DMP1:   LD    B,16      ;contor de octeti in linie
```

```
DMP2:  LD    A,(HL)    ;un octet din zona in A
       CALL  BINHEX    ;conversie in hexa si afisare octet ?
       INC   HL        ;adresa octet urmator
       DEC   DE        ;numar total de octeti
       LD    A,D       ;test daca DE=0
       OR    E
       RET   Z         ;iesire daca DE=0
       DJNZ  DMP2      ;salt daca B > 0
       CALL  CRLF      ;dupa 16 octeti se trece la linie noua
       JR    DMP1
```

Instrucțiunile pentru *deplasare aritmetică*, precum și instrucțiunile de adunare-scădere cu CY pe 16 biți se folosesc în subrutinele aritmetice pentru operații cu întregi de 32 biți și cu numere reale reprezentate în binar virgulă mobilă. Reducerea lungimii acestor subrutine aritmetice se obține și prin realizarea deplasărilor direct în memorie, eliminînd aducerea operanzilor în registre. Exemplu:

Secvență de împărțire prin 2 a unui număr algebric de un cuvînt aflat la adresa din HL.

```
INC   HL     ;adresa octet superior
SRA   (HL)   ;deplasare cu extensie semn octet superior
DEC   HL     ;adresa octet inferior
RR    (HL)   ;deplasare cu CY octet inferior
```

Instrucțiunile pentru *deplasare zecimală* RLD, RRD își găsesc utilizarea în aritmetica cu numere binar-zecimale (BCD); deplasarea la stînga, cu o poziție echivalează, cu o înmulțire prin 10, iar deplasarea la dreapta echivalează cu o împărțire prin 10. Ambele operații sînt necesare la înmulțirea și împărțirea de numere BCD precum și pentru reducerea numerelor zecimale neîntregi la numere întregi.

Exemplu: Împărțirea prin 10 a unui număr BCD, fără semn aflat în memorie la adresa din HL cu lungime dată în B.

```
       XOR   A         ;zero in A
RD1:   RRD             ;deplasare dreapta cu o pozitie
       INC   HL
       DJNZ  RD1
```

Instrucțiunea SBC poate fi folosită pentru comparația la egalitate a două cuvinte (se poziționează indicatorul Z), dar nu se poate folosi pentru comparație la inegalitate, deoarece nu se poziționează indicatorii CY și S.

6.3. Utilizarea directivelor de asamblare

6.3.1. Directive pentru zone de date

Numărul și numele directivelor de asamblare depind de asamblorul folosit, dar directivele uzuale referitoare la date au același nume și semnificație în toate limbajele de asamblare.

Există totuși unele diferențe la numele directivelor utilizate împreună cu mnemonicele Zilog, față de directivele utilizate cu mnemonicele Intel:

INTEL	ZILOG
DS	DEFS
DB	DEFB
DW	DEFW

Macroasamblorul M80 acceptă ambele nume pentru directivele menționate.

Instrucțiunile unui program prelucrează date și produc alte date (numite și rezultate ale prelucrării). Indiferent dacă sînt date inițiale, rezultate intermediare sau rezultate finale, datele necesită memorie.

Zonele de date se rezervă de obicei înainte de începerea execuției programului și au aceeași semnificație pe toată durata de execuție; asemenea date se numesc și date statice sau date alocate static (permanent).

Unele aplicații folosesc și zone de date alocate dinamic, a căror dimensiune și interpretare se pot modifica în cursul execuției programului.

Alocarea statică de memorie pentru date se face cu ajutorul directivei DS (Define Storage):

```
nume: DS n
```

unde „n" este numărul de octeți rezervați. Operandul „n" poate fi în general o expresie evaluată în cursul asamblării dar de obicei este o constantă numerică zecimală.

Unitatea de alocare este octetul, indiferent dacă zona rezervată este structurată pe cuvinte de doi octeți sau pe alte unități de date. Exemplu de rezervare a unui cuvînt pentru memorarea unei adrese:

```
ADR: DS 2
```

Zonele rezervate pentru date separate primesc în general nume distincte sub formă de etichete pentru directivele de rezervare. Exemplu de rezervare pentru 3 variabile de cîte un octet fiecare:

```
ORE: DS 1
MIN: DS 1
SEC: DS 1
```

Este posibilă însă și etichetarea unui bloc de date printr-un singur nume, urmînd ca referirea la componentele acestui bloc să se facă prin adresare relativă de forma următoare:

LDA ORE + 1 în loc de: LDA MIN

Unele zone de date conțin valori constante, iar alte zone cu conținut variabil trebuie inițializate înainte de lansarea programului. Pentru rezervarea și inițializarea simultană a unor zone de date în limbaj de asamblare se folosesc mai multe directive, în funcție de tipul și de lungimea datelor generate:

```
nume: DB b1, b2,...
```

pentru generarea a una sau mai multe valori constante pe cîte un octet fiecare;

```
nume: DB 'text'
```

pentru generarea unui șir de caractere;

```
nume: DW w1, w2,...
```

pentru generarea de valori constante pe cîte un cuvînt fiecare.

În directiva DB se pot folosi împreună ca operanzi atît texte cît și constante numerice. Exemplu:

```
DB 13, 10, 'Eroare', 13, 10, '$'
```

Inițializarea unor zone de memorie cu directive de asamblare este preferată de multe ori inițializării la execuție prin instrucțiuni de memorare a constantelor de inițializare, pentru a permite reducerea lungimii programelor. *Exemplu* de inițializare a 3 variabile de cîte un octet cu zero:

```
ORE:  DB 0
MIN:  DB 0
SEC:  DB 0
```

Trebuie observat însă că inițializarea la execuție nu poate fi înlocuită întotdeauna cu inițializarea la asamblare; de exemplu, atunci cînd inițializarea trebuie refăcută de mai multe ori pe parcursul unui program.

Un caz particular important de inițializare la asamblare este cel al generării unor constante de adresă, destinate unor adresări indirecte, ca în *exemplul* următor:

```
LISTA:   DS  200    ; o listă de octeți
ADRLST:  DW  LISTA  ; adresa listei de octeți
         ...
LHLD ADRLST         ; adresă listă în HL
MOV      A, M       ; un octet din listă în A
```

Dacă este necesară refacerea periodică a adresei de început a listei în cuvîntul ADRLST, atunci se va utiliza secvența de instrucțiuni următoare:

```
LXI    H, LISTA
SHLD   ADRLST
```

Directiva de echivalare EQU este utilizată de obicei pentru a defini *constante simbolice*, prin atribuirea de nume unor constante:

```
CR      EQU 13   ; caracterul de trecere la început de linie
LF      EQU 10   ; caracterul de trecere la linia următoare
BDOS    EQU 5    ; adresa de intrare în BDOS
```

Utilizarea de constante simbolice face programele mai ușor de citit și, dacă este cazul, mai ușor de modificat; de aceea unele adrese de memorie, adresele porturilor de I/E, codurile unor caractere de control neafișabile și alte constante susceptibile de a se modifica la transferul programului pe un alt microsistem trebuie echivalate cu nume simbolice și grupate într-o parte dependentă de mașină.

Directiva EQU poate fi folosită și pentru definirea unor adrese sau unor constante care diferă între ele prin cantități fixe, ca în exemplele următoare:

```
CCP     EQU    0E400H          ; adresa CCP
BDOS    EQU    CCP + 800H      ; adresa BDOS
BIOS    EQU    BDOS + 0E00H    ; adresa BIOS
```

Directiva de stabilire a adresei de încărcare ORG este necesară în programele absolute, dar poate fi folosită și în programele relocabile. Se utilizează pentru plasarea în memorie la anumite adrese a unor secțiuni de cod sau de date neadiacente. Exemplu:

```
BIOS    EQU    0FA00H
        ORG    BIOS
        ...
        ORG    CCP + 8
        ...
```

Nu este necesar ca adresele conținute în directive ORG succesive să fie în ordine crescătoare.

În unele programe se pot întîlni rezervări de memorie realizate cu directivele EQU sau ORG, în loc de a se folosi directiva DS; de exemplu secvența:

```
BUFF     EQU    80H
STACK    EQU    BUFF + 80H
```

este echivalentă cu secvența:

```
         ORG    80H
BUFF:
         ORG    100H
STACK:
         ORG    ....
```

Programele care urmează să fie executate în memorii fixe ROM trebuie să aibă partea de date (modificabilă) net separată de partea de cod și de date constante (nemodificabilă la execuție). Pentru asemenea programe sînt destinate directivele CSEG și DSEG; toate secvențele de directive precedate de directiva DSEG sînt colectate într-o *secțiune unică de date* și toate secvențele de instrucțiuni și de directive precedate de directiva CSEG sînt grupate într-o *secțiune unică de cod,* indiferent de pozițiile unde apar scrise aceste secvențe în program (adresele se alocă succesiv în cadrul fiecărei secțiuni și nu în cadrul întregului program). De *exemplu,* secvența următoare:

```
        CSEG
START : ...
        DSEG
D1:     DS      10    ; zone de date în RAM
        CSEG
S1:     ...           ; secvență în ROM
        DSEG
D2:     DS      20    ; alte date în RAM
        CSEG
S2:     ...           ; secvență în ROM
        END     START
```

este echivalentă cu alocare a memoriei cu secvența următoare:

```
        CSEG          ; în ROM
START : ...
S1:     ...
S2:     ...
        DSEG          ; în RAM
D1:     DS      10
D2:     DS      20
        END     START
```

Adresele de încărcare în memorie pentru secțiunea de date și pentru secțiunea de program sînt controlabile la asamblare (prin directive ORG) și la linkeditare (prin opțiunile /P și /D la L80).

Pentru programele încărcate în întregime în memoria RAM (la microcalculatoare) directivele CSEG și DSEG pot să nu fie utilizate deoarece în mod implicit asamblorul consideră tot programul ca o secțiune de cod (o secțiune de cod poate conține și date sau numai date, deoarece asamblorul nu verifică conținutul unei secțiuni).

6.3.2. Directive pentru programe modulare

Împărțirea unui program în mai multe module, asamblate separat,

poate fi necesară sau utilă în mai multe situații:

— în cazul programelor mari, dezvoltate și testate treptat, prin acumularea de noi funcții;

— în cazul subrutinelor destinate a fi introduse într-o bibliotecă de subprograme;

— în cazul subprogramelor scrise în limbaj de asamblare, dar utilizate din alte limbaje de programare.

Un modul de program poate primi un nume prin directiva NAME sau TITLE.

Comunicarea de date între module asamblate separat se poate face:

— prin registrele procesorului;
— prin anumite adrese fixe de memorie;
— prin nume simbolice globale (variabile globale);
— prin blocuri de memorie comune.

Referințele simbolice între module diferite pot fi:

— apeluri de subrutine dintr-un alt modul;
— referiri la date dintr-un alt modul.

Numele simbolice (etichete sau constante simbolice) utilizate în mai multe module asamblate separat se numesc simboluri globale și pot fi de două tipuri:

— *puncte de intrare*, definite în modulul curent, dar folosite (și) în alte module;

— *referințe externe*, folosite în modulul curent, dar definite în alte module.

Ambele tipuri trebuie declarate asamblorului; absența acestor declarații produce erori la editarea de legături a modulelor respective (eroarea apare după căutarea în biblioteca implicită și deci este semnalată ca absența unui modul din bibliotecă).

Etichetele puncte de intrare pot fi declarate în cazul asamblorului M80 astfel:

— printr-o directivă PUBLIC;
— printr-o directivă ENTRY;
— prin folosirea a două caractere':' după etichetă. Exemple:

```
    PUBLIC  MOVE, DRAW
sau :
    ENTRY   MOVE, DRAW
sau :
    MOVE:: ...
    DRAW:: ...
```

Pot exista și constante simbolice globale, definite numai prin directivele PUBLIC sau ENTRY.

Numele simbolice externe pot fi declarate:

— printr-o directivă EXT;

— printr-o directivă EXTRN;

— prin folosirea a două caractere '#' după numele respectiv (convenție M80). Exemple:

EXT MOVE, DRAW

sau

EXTRN MOVE, DRAW

sau

CALL MOVE##
CALL DRAW##

Pentru a exemplifica referirile simbolice între module atît la instrucțiuni, cît și la date, vom folosi următoarele două module simple: un program principal și o subrutină, fiecare din ele afișează valori atribuite în cealaltă unitate de program.

Un fișier sursă conține *programul principal*:

```
;program principal
        ENTRY      V1
        EXTRN      SUB.V2
MAIN    MVI   A, 1       .initializare V1
        STA   V1
        CALL  SUB        .SUB scrie pe V1 si initializeaza pe V2
        LDA   V2         .scrie valoarea lui V2
        MOV   E.A
        MVI   C.2        .cod functie BDOS
        CALL  5          .apel sistem
        JMP   0          .terminare program
V1:     DS    1
        END   MAIN
```

Alt fișier conține *subrutina*:

```
;subrutina SUB
        EXTRN      V1
        ENTRY      SUB,V2
SUB:    LDA   V1         ;scrie valoarea lui V1
        MOV   E,A
        MVI   C,2        ;cod functie BDOS
        CALL  5          ;apel BDOS
        MVI   A,'2'      ;initializare V2
        STA   V2
        RET
V2:     DS    1
        END
```

Comunicarea între module prin *blocuri comune* necesită folosirea directivei COMMON cu același nume de bloc comun în toate modulele care trebuie să aibă acces la memoria comună respectivă.

O secțiune de date comună începe la o directivă COMMON și se termină la prima directivă COMMON, CSEG sau DSEG întîlnită.

Etichetele din cadrul unei secţiuni comune nu trebuie să fie identice pentru toate modulele, deoarece ele reprezintă doar o descriere, un punct de vedere al unui modul asupra unei secţiuni comune. Lungimea tuturor secţiunilor comune cu acelaşi nume este de obicei egală; in orice caz lungimea blocului comun este determinată de prima directivă COMMON întîlnită de linkeditor (la L80), de regulă în programul principal, şi nu trebuie să urmeze altă secţiune mai mare cu acelaşi nume.

De observat deosebirea dintre directivele COMMON şi directivele CSEG, DSEG: datele din secţiunile de comun cu acelaşi nume se suprapun într-o aceeaşi zonă de memorie, iar datele din secţiunile de date sau de program se concatenează la adrese succesive.

Pentru exemplificarea comunicării de date prin secţiuni comune vom relua exemplul celor două module care afişează variabile iniţializate de celălalt modul.

;program principal (fişier separat)

```
        COMMON    /C1/
X:      DS   1
        COMMON    /C2/
Y:      DS   1
        CSEG
        EXT  SUB
MAIN:   MVI  A,'1'        ;initializare X
        STA  X
        CALL SUB          ;SUB scrie pe X si init. pe Y
        LDA  Y            ;scrie pe Y
        MOV  E,A
        MVI  C,2          ;cod functie BDOS
        CALL 5            ;apel sistem
        JMP  0
        END  MAIN

;subrutina SUB (fisier separat)
        COMMON    /C1/
X:      DS   1
        COMMON    /C2/
Y:      DS   1
        CSEG
        ENTRY     SUB
SUB:    LDA  X            ;scrie pe X
        MOV  E,A
        MVI  C,2          ;cod functie BDOS
        CALL 5            ;apel sistem
        MVI  A,'2'
        STA  Y
        RET
        END
```

În incheiere facem observaţia că mai toate compilatoarele limbajelor uzuale (Fortran, Pascal, C) generează secţiuni comune pentru

variabilele globale (comune) și că toate numele de subprograme sînt generate automat ca puncte de intrare la definirea lor și ca referințe externe la apelarea lor.

6.3.3. Directive pentru macroinstrucțiuni

O macroinstrucțiune înlocuiește o secvență de instrucțiuni sau de directive printr-un singur nume. Secvența prin care se expandează o macroinstrucțiune poate să fie fixă, dar mai des ea conține parametri variabili, înlocuiți cu argumentele macroinstrucțiunii. Mai mult decît atît, chiar lungimea și conținutul secvenței pot fi controlate prin argumente ale macroinstrucțiunii.

Utilizarea de macroinstrucțiuni este o tehnică de programare folosită mai puțin în sistemul CP/M decît în alte sisteme de operare, în sensul că nu există macroinstrucțiuni predefinite pentru apeluri de funcții sistem.

Principala utilizare a macroinstrucțiunilor este aceea de a prescurta secvențele de apelare a unor subrutine cu mai mulți parametri, dar în CP/M subrutinele de sistem (funcțiile BDOS) au numai unul sau doi parametri și în general nu este încurajată folosirea de subrutine cu multe argumente.

Pentru anumite aplicații fiecare utilizator își poate defini o colecție de macroinstrucțiuni (o bibliotecă de macrodefiniții) pentru apelarea repetată a unor subrutine cu mai mulți parametri.

O macrodefiniție este delimitată de directiva:

```
nume        MACRO        arg1, arg2,...
```

la început și de directiva

```
            ENDM
```

la sfîrșitul secvenței ce constituie macrodefiniția. Exemplu: Macroinstrucțiune pentru transferul unui șir de octeți de la o adresă sursă la o adresă destinație pe o lungime dată, pentru 8080:

```
STRCPY  MACRO      SRC,DST,LNG
        LXI   H,SRC
        LXI   D,DST
        MVI   B,LNG
        CALL  MOVE
        ENDM
```

Utilizarea acestei macroinstrucțiuni se va face printr-un macroapel de forma:

```
STRCPY Z1, Z2, 80
```

Această linie este înlocuită (expandată) de asamblor prin secvența următoare:

```
LXI H, Z1
LXI D, Z2
MVI B, 80
CALL MOVE
```

Macroinstrucțiunile nu constituie o alternativă pentru subrutine, deoarece definirea de subrutine pentru operații solicitate de mai multe ori într-un program conduce la reducerea lungimii programului executabil, iar definirea de macroinstrucțiuni conduce numai la reducerea lungimii programului sursă.

O *bibliotecă de macroinstrucțiuni* este un fișier sursă care conține numai macrodefiniții și poate fi creat sau modificat cu orice editor de texte.

La asamblarea unui program care folosește macroinstrucțiuni dintr-o bibliotecă trebuie inclus în asamblare și fișieul bibliotecă printr-o directivă MACLIB (la asamblorul MAC) sau printr-o directivă #INCLUDE (la asamblorul M80). Această soluție este preferabilă concatenării celor două fișiere sursă într-un singur fișier (cu PIP) înainte de asamblare, deoarece programul este mereu modificat în cursul punerii sale la punct, iar biblioteca rămîne în general aceeași.

Directivele MACLIB sau #INCLUDE se folosesc la începutul programului sau înainte de primul apel de macro din bibliotecă.

Directivele #INCLUDE și MACLIB se pot folosi și pentru alte fișiere sursă decît biblioteci macro, în scopul evitării editării de fișiere sursă prea mari la dezvoltarea de programe.

Directivele de *asamblare condiționată* și de *asamblare repetată* (IF, REPT, IRP) se utilizează fie în cadrul unor macrodefiniții, fie în programe generale, parametrizate, care urmează a fi adaptate prin parametrii de asamblare la anumite condiții de lucru specifice sau la anumite configurații hardware.

Exemplu de folosirea directivei **IFB/IFNB** într-o macro de afișare caracter la consolă :

```
PUTC    MACRO     CHAR
        IFNB      CHAR     ;daca exista argument macro
        MVI  E,CHAR        ;atunci se include MVI E in macro
        ENDIF              ;altfel nu mai apare MVI E in macro
        MVI  C,2
        CALL 5
        ENDM
```

Utilizată cu argument, macroinstrucțiunea PUTC generează și instrucțiunea de încărcare în registrul E a valorii caracterului de afișat; fără argument PUTC se va folosi atunci cînd registrul E se încarcă înainte de apelarea macroinstrucțiunii.

Sînt posibile și macroinstrucțiuni fără argumente. Exemplu: Macroinstrucțiune care generează cod mașină pentru o instrucțiune Z80 și care poate fi asamblată cu un asamblor de Z80 sau de 8080:

```
LDIA    MACRO
        IF      Z80
        LD      I, A
```

```
        ELSE
        DB      0EDH, 047H   ;cod intern instrucţiune LDIA
        ENDIF
```

Pentru a ilustra un program general parametrizat vom da ca *exemplu* un mic fragment dintr-un program BIOS pentru o maşină CP/M care poate lucra cu discuri de 8" sau de 5".

```
MINI    SET     FALSE       ;parametru pentru tip disc flexibil
        ...

;tabela de parametrii pentru comenzi 8272

        IF      MINI
N       DB      1           ; dublă densitate
EOT     DB      16          ; număr de sectoare pe pistă
GPL     DB      0EH         ; mărime interval între sectoare
DTL     DB      255         ; lungime sector (date)
        ELSE
N       DB      0           ; simplă densitate
EOT     DB      26          ; număr de sectoare pe pistă
GPL     DB      07          ; mărime interval
DTL     DB      128         ; lungime sector (date)
        ENDIF
```

Nu vom insista aici asupra celorlalte facilităţi oferite de asamblorul M80 pentru definirea de macroinstrucţiuni, considerînd că utilizarea intensă de macroinstrucţiuni poate conduce la o mărire nejustificată a timpului de asamblare în condiţiile discurilor flexibile utilizate pe microcalculatoare.

7. INTRĂRI-IEȘIRI ÎN LIMBAJ MAȘINĂ ÎN CP/M

Programarea operațiilor de intrare-ieșire în limbaj mașină se poate face la mai multe niveluri, în funcție de echipamentele pe care se folosesc programele scrise și de specificul acestor programe.

Nivelul superior de exprimare a operațiilor de I/E folosește funcțiile de I/E, puse la dispoziție de către sistemul de operare, numite funcții BDOS în CP/M.

Marea majoritate a programelor utilizate sub sistemul CP/M au fost scrise în limbaj de asamblare cu funcții BDOS de intrare-ieșire, ceea ce le asigură universalitate; de aici posibilitatea schimbului de programe între utilizatori și deci reducerea considerabilă a efortului de programare la fiecare instalație în parte.

Condiția care trebuie respectată de programe pentru a fi portabile este utilizarea exclusivă a funcțiilor oferite de sistem pentru programarea operațiilor de intrare-ieșire și evitarea utilizării unor adrese de memorit. de porturi de I/E sau a unor alți parametri dependenți de echipamene,

Un număr relativ redus de programe portabile folosesc și funcții BIOS de intrare-ieșire, deoarece trebuie să folosească discul la nivel de pistă și de sector și nu la nivel de fișier (de exemplu, utilitarele SYSGEN, DU, POWER).

Dar nu orice program care folosește funcții sistem este un program portabil, adică imediat utilizabil și pe o altă mașină CP/M, fără modificări sau cu un minim de modificări în programul obiect, fără a dispune de forma sursă a programului.

O serie de programe scrise numai cu funcții BDOS, dar care funcționează cu consola în mod ecran (Wordstar, Datastar, dBASE, Multiplan, Turbo-Pascal) trebuie totuși adaptate la particularitățile video-terminalului folosit printr-o operație de ,,instalare'' (de configurare) realizată fie manual, prin modificarea unor octeți din program la adresele indicate, fie automat, printr-un program de instalare care primește parametrii necesari adaptării printr-un dialog cu operatorul la consolă.

Nivelul inferior de programare a operațiilor de I/E folosește direct instrucțiunile de I/E ale procesorului și eventual instrucțiunile referitoare la sistemul de întreruperi.

Acest nivel reprezintă singura posibilitate pentru programele destinate unor echipamente fără sistem de operare și pentru dispozitivele speciale de I/E, netratate de către sistem.

Uneori programele destinate unor echipamente specializate pot utiliza subrutine de I/E din memoria permanentă, cunoscînd adresele acestor subrutine și convenția de comunicare a datelor.

7.1. Funcții BDOS de intrare-ieșire

Forma de apelare a funcțiilor BDOS

Funcțiile sistem CP/M sînt rutine din modulul BDOS, apelate prin secvențe de două sau de trei instrucțiuni din cadrul programelor scrise de utilizatori.

Toate rutinele BDOS au un punct de intrare unic, care este adresa 5 din zona sistem de la începutul memoriei, iar selectarea rutinei dorite se face printr-un cod de funcție transmis prin registrul C.

Rutinele BDOS necesită în general un parametru de intrare de un octet sau de un cuvînt, care trebuie transmis prin registrul E sau prin perechea de registre DE; rezultatul executării unei funcții BDOS apare în registrul A și uneori în registrele HL.

Executarea unei funcții BDOS modifică toate registrele procesorului, ceea ce implică responsabilitatea programatorului pentru salvarea și refacerea registrelor din program, care trebuie să rămînă neschimbate după executarea unei funcții sistem.

Exemplu de apelare a unei funcții BDOS fără parametru:

```
MVI C,1; cod funcție „citește caracter la CON"
CALL 5; apel sistem
...    ; registrul A conține caracterul citit
```

Exemplu de apelare funcție BDOS cu parametru de un **octet**:

```
MVI C, 2; cod funcție „afișare caracter la CON"
MVI E, '#'; valoare caracter afișat
CALL     5; apel sistem (nu există rezultat)
```

Exemplu de apelare funcție BDOS cu parametru de un **cuvînt**:

```
MVI C, 9; cod funcție „afișare text"
LXI D, TEXT; adresa text în DE
CALL 5; apel sistem (nu există rezultat).
```

Funcțiile BDOS apelate de mai multe ori pot fi programate ca

subrutine de intrare-ieşire. Exemple: subrutine pentru deschiderea şi închiderea unui fişier:

```
OPEN:  MVI C, 15; cod funcţie „deschidere fişier"
       JMP 5
CLOSE: MVI C, 16; cod funcţie „închidere fişier"
       JMP 5
```

Revenirea din BDOS se face cu o instrucţiune RET la adresa din vîrful stivei, deci la adresa imediat următoare instrucţiunii CALL, care face apel la subrutina de I/E.

Salvarea şi refacerea registrelor programului care face apelul se pot face în afara subrutinei sau în subrutina de I/E. *Exemplu* de subrutină pentru afişarea la consolă a unui caracter dat în registrul E, cu salvarea registrelor:

```
CO: PUSH B
    PUSH D
    PUSH H
    MVI  C, 2
    CALL    5
    POP  H
    POP  D
    POP  B
    RET
```

Secvenţele de apelare a subrutinelor de I/E sau a funcţiilor BDOS pot fi eventual definite ca macroinstrucţiuni, pentru a face programele mai compacte şi mai uşor de urmărit. *Exemplu* de macroinstrucţiune pentru afişarea unui text la consolă:

```
PRINT  MACRO TXTADR
       LXI D, TXTADR
       MVI   C,9
       CALL  5
       ENDM
```

Utilizarea acestei macroinstrucţiuni pentru a scrie diferite mesaje arată astfel:

```
PRINT EROPEN    ;scrie mesaj de la adresa EROPEN
...
PRINT ERREAD    ;scrie mesaj de la adresa ERREAD
```

Programatorii care preferă să folosească macroinstrucţiuni pentru a face apel la funcţiile sistem de I/E pot să creeze o bibliotecă cu definiţiile acestor macroinstrucţiuni şi apoi să folosească în programe o directivă de includere la asamblare a fişierului bibliotecă cu macroinstrucţiuni:

```
# include BDOSLIB    ;pentru asamblorul M80
MACLIB  BDOSLIB      ;pentru asamblorul MAC
```

7.1.1. Funcţii de intrare-ieşire pentru dispozitive standard

1 = *Citire de la consolă* (Console Input)

Se citeşte un caracter de la consolă în registrul A; fie se aşteaptă tastarea caracterului, fie se preia un caracter introdus anterior executării acestei funcţii. Caracterul citit este afişat în ecou la consolă, inclusiv caracterele de control ⟨CR⟩, ⟨LF⟩ şi ⟨BS⟩. Caracterele TAB sînt înlocuite cu spaţii pînă se ajunge la următoarea limită de zonă de 8 poziţii. Această funcţie interpretează caracterele de control CTRL/C CTRL/P, CTRL/Q şi CTRL/S.

2 = *Afişare la consolă* (Console Output)

Se trimite la ecranul consolei, pentru afişare, caracterul din registrul E. Se expandează caracterele TAB şi se interpretează CTRL/P (ecou la imprimantă), CTRL/S (oprire/pornire afişare) şi CTRL/Q (repornire afişare).

3 = *Introducere de la dispozitivul de citire* (Reader Input)

Se citeşte un caracter în registrul A de la dispozitivul logic de intrare al sistemului (RDR), cu eventuală aşteptare.

4 = *Ieşire la dispozitivul de perforare* (Punch Output)

Se trimite la dispozitivul perforator al sistemului (PUN) caracterul din registrul E.

5 = *Ieşire la dispozitivul de listare* (List Output)

Se trimite pentru imprimare caracterul din registrul E la dispozitivul de listare al sistemului (LST).

6 = *Intrare/ieşire directă prin consolă* (Direct Console I/O)

Dacă registrul E conţine OFFH, atunci se citeşte un caracter de la consolă în registrul A fără interpretare; dacă registrul E conţine o altă valoare, atunci se afişează la consolă caracterul al cărui cod ASCII se află în registrul E (fără interpretarea caracterelor de control). Dacă nu s-a introdus nici un caracter de la consolă, iar în registrul E a fost FFH, atunci registrul A va conţine 0, pentru a indica această situaţie (nu se aşteaptă tastarea caracterului).

9 = *Afişare text la consolă* (Print String)

Se afişează la consolă şirul de caractere care începe la adresa din registrele DE şi care se termină cu caracterul „$". Se expandează caracterele TAB şi se interpretează caracterele CTRL/P, CTRL/Q CTRL/S.

10 = *Citire linie de la consolă* (Read Console Buffer)

Se citeşte o linie de la consolă în memorie, începînd de la adresa aflată în registrele DE plus 2. În octetul de la adresa din DE se află numărul maxim de caractere din linie (m), pus de programator. În octetul imediat următor sistemul pune numărul de caractere citite efectiv, iar în continuare caracterele introduse de la consolă. O linie se termină la introducerea unui caracter terminator (⟨CR⟩, ⟨LF⟩, ↑E), sau după

introducerea a m-2 caractere. Caracterul terminator de linie nu este depus în zona tampon, iar zona tampon nu este inițializată automat înainte de citire. Linia citită este afișată în ecou la consolă. În cursul introducerii unei linii sînt interpretate următoarele caractere de control:
DEL (RUBOUT) șterge ultimul caracter introdus și îl reafișează
BS (Backspace) șterge ultimul caracter introdus cu un blanc
CTRL/U șterge linia curentă și avansează la linia următoare
CTRL/X șterge linia curentă și trece la începutul liniei
CTRL/R reafișează linia curentă
CTRL/C reinițializare sistem

11 = *Citire stare consolă* (Get Console Status)

Se încarcă în registrul A un octet de stare care are valoarea zero, dacă nu s-a introdus nici un caracter, și o valoare diferită de zero, dacă s-a tastat o clapă.

O consolă cu ecran are o serie de posibilități suplimentare față de o consolă cu imprimare, iar multe programe folosesc aceste posibilități.

În primul rînd trebuie observat că sînt posibile două moduri de folosire a ecranului: *modul defilare* („scroll") și *modul pagină* sau bloc („page"); în modul defilare se afișează mereu în linia de jos a ecranului (linia 23 de obicei), iar liniile anterioare sînt mutate în sus („defilează") pierzîndu-se linia de sus care iese din ecran; în modul pagină se începe afișarea din linia de sus și se coboară în jos la fiecare linie nouă, pînă cînd s-a ajuns în linia de jos, cînd se reia afișarea din linia de sus (liniile noi se afișează peste liniile anterioare, deoarece ecranul nu se șterge automat la schimbarea paginii pe ecran).

Modul pagină se folosește în regim grafic, unde nu se acceptă defilarea automată a imaginii de pe ecran; modul defilare este modul de lucru curent la afișarea de texte alfanumerice.

Modul ecran („screen") sau modul video este folosit tot pentru afișarea de texte alfanumerice, dar nu pe linii succesive, ci pe orice linie și în orice poziție din linie. Principalele facilități oferite în mod ecran sînt:

— poziționarea cursorului pe ecran în orice poziție de caracter, ecranul fiind considerat ca o matrice de 24 de linii și 80 de coloane, cu poziția 0,0 în colțul din stînga sus;

— deplasare cursor cu o poziție în fiecare din cele 4 direcții;

— ștergere rapidă a întregului ecran;

— ștergerea caracterelor din linia curentă aflate la dreapta cursorului;

— ștergerea liniilor de sub linia curentă;

— comutarea între două intensități de afișare sau între video normal și video invers, între oricare două caractere de pe ecran, pentru scoaterea în evidență a unor texte sau semne speciale.

Numărul facilităţilor în mod ecran şi modul în care se comandă obţinerea acestor efecte diferă de la un video-terminal la altul sau de la un calculator la altul; de asemenea pot fi diferite dimensiunile matricei de caractere de pe ecran.

Pentru anumite comenzi se foloseşte cîte un singur caracter de control (cu cod ASCII mai mic de 20H), iar pentru comanda de poziţionare şi alte comenzi se folosesc secvenţe de caractere, care încep cu un caracter de control special (de obicei caracterul „Escape", ESC=27=1BH).

Interpretarea acestor caractere şi secvenţe de control se face de către rutina BIOS de afişare a unui caracter sau de către rutina din monitorul rezident, apelată de rutina BIOS sau de către circuitul controlor de ecran.

Transmiterea comenzilor necesare lucrului în mod ecran se poate face cu funcţii BDOS de scriere la consolă.

Iată principalele caractere şi secvenţe de control în mod ecran folosite la Felix M118, TPD şi JUNIOR:

OEH comută pe video invers
OFH comută pe video normal
16H şterge caractere după cursor din linia curentă
18H şterge ecran
19H poziţionare cursor în colţul din stînga sus
1BH, 31H, x+20H, y+20H poziţionare cursor în linia y, poziţia x
1BH, 32H Comutare din mod defilare în mod pagină şi invers
1BH, 49H şterge ecran şi stabileşte mod defilare

La poziţionarea cursorului x reprezintă numărul de caracter în linie, iar y numărul de linie pe ecran; $x = 0$ corespunde primului caracter din linie, iar $y = 0$ corespunde primei linii de sus (linia de jos are de obicei numărul 23).

7.1.2. Alte funcţii BDOS

0 = *Reiniţializare sistem* (System Reset)

Se intră în BIOS, unde se fac toate iniţializările de la pornirea sistemului, se stabileşte discul A ca disc implicit, se sare în CCP, care afişează promptul „>", şi aşteaptă comenzi.

7 = *Citirea octetului de I/E* (Get I/O byte)

Se încarcă în registrul A octetul de I/E (de la adresa 3).

8 = *Scrierea octetului de I/E* (Set I/O byte).

Se memorează în octetul de I/E (de la adresa 3) valoarea din registrul E.

12 = *Citire număr versiune* (Return Version Number)

Se încarcă în registrul H un octet 0 şi în registrul L un octet cu valoarea 20H pentru versiunea 2.0.

13 = *Reinițializare discuri* (Reset Disk System)

Se readuc toate discurile în starea R/W (read/write), se stabilește discul A ca disc implicit și se stabilește adresa implicită de citire (scriere) de pe disc la valoarea 80H.

14 = *Selectare disc* (Select Disk)

Se stabilește ca disc implicit de lucru unitatea al cărui număr se dă în registrul E, cu convenția E = 0 pentru discul A, E = 1 pentru discul B. Se citește directorul discului din unitatea selectată și se creează în memorie tabela de alocare.

24 = *Obținere vector discuri selectate* (Get Login Vector)

Se încarcă în registrele HL un vector de 16 biți, cu octetul inferior în registrul L, în care fiecare bit corespunde unei unități de discuri (bitul 0 din L pentru discul A, bitul 1, pentru discul B etc.). Un bit 1 arată că unitatea respectivă a fost selectată, iar un bit 0 că unitatea nu a fost selectată. Selecția se poate face fie prin funcția 14, fie printr-o funcție de deschidere de fișier (15, 17, 18 sau 22). Registrul A conține și el valoarea din registrul L.

25 = *Obținere disc curent* (Return Current Disk)

Se încarcă în registrul A numărul discului curent: 0 pentru discul A, 1 pentru discul B etc.

27 = *Obținere adresă vector de alocare* (Get Alloc Address)

Se încarcă în registrele HL adresa vectorului de alocare pentru discul curent. În BIOS se păstrează cîte un vector de alocare pentru fiecare disc conectat în sistem, care arată starea de ocupare a discului.

28 = *Protejare disc la scriere* (Write Protect Disk)

Se protejează la scriere discul curent selectat, protecție care rămîne pînă la o nouă inițializare a sistemului.

29 = *Obține vector discuri protejate* (Get Read-Only Vector)

Se încarcă în registrele HL un vector de 16 biți care arată starea discurilor cuplate la sistem: un bit = 1 arată că discul respectiv este protejat la scriere (Read-Only). Bitul 0 din registrul L corespunde discului A, bitul 1 discului B, etc.

7.1.3. Exemple de utilizare a funcțiilor BDOS de I/E

1) Exemplu pentru funcția 1: Secvență de afișare a conținutului unui fișier, sector cu sector, cu oprire după fiecare sector afișat; afișarea sectorului următor se face după apăsarea unei taste oarecare; CTRL/C termină afișarea.

```
CICLU:  CALL RDSEQ     ;citeste sectorul urmator
        CALL DISPL     ;afisare sector la consola
        MVI  C,1       ;asteapta tastare caracter la consola
        CALL 5
        JMP  CICLU     ;reluare ciclu dupa tastare caracter
```

2) Exemplu pentru funcția 11: Afișarea continuă a conținutului unui fișier cu posibilitatea de terminare a afișării prin tastarea caracterului DEL.

```
CICLU:  CALL RDSEQ     ;citeste sectorul urmator
        CALL DISPL     ;afisare sector la consola
        MVI  C,11      ;s-a tastat caracter ?
        CALL 5
        ORA  A         ;pozitionare indicatori
        JZ   CICLU     ;repeta daca nu s-a tastat
        MVI  C,1       ;citeste caracterul tastat
        CALL 5
        CPI  7FH       ;a fost DEL ?
        JZ   0         ;terminare afisare daca DEL
```

3) Exemple pentru funcția 2: Subrutine de ștergere ecran
Dependent de mașină (numai la M118, TPD, JUNIOR):

```
CLEAR   EQU  18H       ;caracter de control special
CLRSCR: MVI  E,CLEAR   ;trimite caracter de stergere ecran
        CALL CO        ;Apel subrutina de afisare caracter
        RET
```

Independent de mașină (dar mult mai încet):

```
CR      EQU  13
LF      EQU  10
CLRSCR: MVI  E,CR      ;trimite caracter Return
        CALL CO
        MVI  B,23      ;numar de linii pe ecran
CLR1:   MVI  E,LF      ;trimite caracter Line Feed
        CALL CO
        DCR  B
        JNZ  CICLU
        RET
```

4) Exemplu pentru funcția 9: Secvență de afișare a unei întrebări și citire răspuns:

```
; afisare intrebare
        LXI  D,INTREB
        MVI  C,9
        CALL 5
;citire raspuns de un caracter
        MVI  C,1
        CALL 5
        CPI  'D'
        JZ   DA
  NU:   ...
  INTREB: DB 'RASPUNDETI PRI' DA SAU NU (D/N)$'
```

5) Exemplu pentru funcția 6: Secvență de citire și interpretare a oricărui caracter:

```
citire caracter fara filtrare de catre sistem
CDC:    MVI  E,OFFH      ;cod citire
        MVI  C,6
        CALL 5
        ORA  A           ;pentru testare reg.A
        JZ   CDC         ;asteapta introducere caracter
interpretare caracter
        CPI  ' '         ;caracter de control ?
        JNC  GRAF        ;salt daca caracter grafic
        PUSH PSW         ;salvare caracter citit
        MVI  E,          ;afiseaza o sageata verticala
        CALL CO
        POP  PSW         ;refacere caracter citit in A
        ADI  40H         ;transforma in caracter afisabil
        MOV  E,A         ; pentru afisare
        CALL CO
```

6) Exemplu pentru funcția 10: Citirea unei linii de la consolă cu introducerea unui octet zero la sfîrșitul liniei:

```
LXI  D,BUF     ;adresa buffer de citire
MVI  C,10      ;cod functie de citire linie
CALL 5         ;citire sir in BUF+2
LXI  H,BUF+1   ;
MOV  A,M       ;numar de caractere citite in A
INX  H         ;HL=adresa primului caracter
CALL ADHL      ;calculeaza adresa dupa ultimul caracter
MVI  M,0       ;pune un zero dupa ultimul caracter
```

7) Exemplu pentru funcțiile 7 și 8: Secvență de realizare a asocierii LST:=UL1: prin modificare octet de I/E:

```
MVI  C,7       ;citeste octet de I/E
CALL 5
ANI  3FH       ;anuleaza bitii 6 si 7
ORI  OCOH      ;pune bitii 6 si 7 pe 1
MOV  E,A       ;pentru functia 8
MVI  C,8
CALL 5         ;scrie octet de I/E
```

8) Exemplu pentru funcția 14: Fragment dintr-un program de copiere fișiere cu schimbare disc, în unitatea A:

```
LXI  D,MESAJ   ;scrie mesaj
MVI  C,9
CALL 5
```

```
MVI  C,1        ;asteapta apasare pe CR
CALL 5
CPI  13
JNZ  WT
MVI  E,0        ;selectie disc A
MVI  C,14
CALL 5
```

7.2. Funcţii BDOS pentru fişiere disc

7.2.1. Funcţii BDOS la nivel de fişier

Pentru fiecare fişier disc utilizat într-un program este necesar un *bloc de control* asociat fişierului („File Control Block" = FCB) în lungime de 36 de octeţi. Acest bloc (numit în continuare pe scurt FCB) conţine numele şi extensia fişierului, precum şi alte informaţii utilizate de sistem pentru accesul la fişier.

Majoritatea funcţiilor BDOS de lucru cu fişiere necesită ca parametru în registrele DE adresa blocului FCB.

Blocul de control este rezervat şi completat parţial de programator (cu nume, extensie şi codul unităţii de disc) şi în rest completat de sistem (de BDOS). Primii 32 de octeţi din FCB sînt aproape identici cu o etichetă de fişier.

La deschiderea unui fişier existent se încarcă în FCB prima etichetă de fişier memorată pe disc, urmînd ca apoi să se încarce şi celelalte etichete (pentru fişiere mai mari de 16 koct). La crearea unui nou fişier se completează în memorie blocul de control fişier, care ulterior se scrie pe disc la închiderea fişierului. De fapt se iniţializează eticheta cu numele fişierului pe disc, chiar de la deschiderea unui fişier nou, iar lista blocurilor ocupate se scrie la închiderea fişierului.

Reamintim că procesorul de comenzi consolă (CCP) foloseşte un bloc de control implicit, aflat la adresa 5CH, unde depune numele (şi ex- (tensia fişierului care apare ca parametru al comenzii. Această zonă de memorie poate fi folosită ca FCB de către programele scrise de utilizatori, chiar dacă comanda de lansare a unui asemenea program nu foloseşte parametri.

Programatorul are sarcina să rezerve şi să iniţializeze un bloc de control fişier pentru fiecare fişier disc utilizat în program; iniţializarea FCB-ului se face cu numele şi extensia în octeţii 1—11 şi cu zerouri binare în ceilalţi octeţi (eventual octetul 0 cu altă valoare). Mai exact, este suficient să se iniţializeze cu zero octetul 32 din FCB (număr sector în fişier) pentru acces secvenţial la fişier. FCB-ul implicit de la adresa 5CH este iniţializat de către CCP în mod corespunzător.

La terminarea cu succes a operaţiilor de lucru cu fişiere (deschidere, închidere) registrul A conţine un cod „director" cu valori între 0 şi 3,

care reprezintă numărul etichetei fișierului în cadrul unui sector din directorul discului. Acest cod poate fi utilizat astfel: se înmulțește codul director cu 32 (lungimea unei etichete), se adună la adresa zonei buffer de I/E și se obține adresa de memorie la care se află eticheta citită.

15 = *Deschide fișier existent* (Open File)

Se folosește pentru căutarea unui fișier cu nume neambiguu în tabela de fișiere a discului selectat de octetul 0 din FCB, în vederea pregătirii fișierului pentru exploatare. Dacă nu se găsește eticheta fișierului cu nume dat în FCB, atunci se pune în registrul A valoarea 0FFH, cu semnificația „fișier inexistent". Dacă se găsește eticheta fișierului indicat, atunci se citește în FCB lista blocurilor ocupate de fișier, se marchează fișierul ca deschis (octetul 14) și se încarcă în registrul A codul director.

16 = *Închide fișier* (Close File)

Se folosește la terminarea prelucrării unui fișier (deschis prin funcțiile 15 sau 22) pentru scrierea în eticheta de fișier a noului conținut din FCB. Dacă fișierul a fost doar citit, operația poate lipsi, deoarece nu s-a modificat eticheta de fișier. Registrul A conține 0FFH, dacă numele de fișier indicat nu a fost găsit în tabela de fișiere a discului selectat.

17 = *Căutarea primului fișier* (Search for First)

Se folosește pentru căutarea primului fișier dintr-un grup de fișiere desemnat printr-un nume ambiguu, ce conține caractere „?" în nume sau în extensie. Căutarea se face pe discul selectat prin octetul 0 din FCB. Registrul A conține 0FFH dacă nu există nici un fișier al cărui nume să se potrivească. Dacă octetul 0 din FCB conține un caracter „?", atunci se caută primul element al tabelei de etichete fișier de pe discul implicit, indiferent dacă este un element liber sau o etichetă de fișier. Această convenție se aplică și pentru funcția 18.

18 = *Căutarea următorului fișier* (Search for next)

Se folosește după funcția 17, pentru găsirea celorlalte fișiere indicate de numele ambiguu, deoarece căutarea continuă de unde s-a găsit eticheta precedentă. Registrul A va conține 0FFH cînd nu mai există nici un fișier al cărui nume să fie un caz particular al numelui ambiguu.

19 = *Șterge fișier* (Delete File)

Se folosește pentru marcarea ca fișier șters a fișierului desemnat de FCB din tabela de fișiere a discului selectat (tot prin FCB). Se pot utiliza și nume ambigue de fișiere. Registrul A conține 0FFH, dacă nu există nici un fișier cu numele și tipul indicat de FCB. Ștergerea unui fișier în CP/M se face prin modificarea primului octet din eticheta de fișier.

22 = *Deschide fișier pentru creare* (Make File)

Se folosește pentru crearea unui nou fișier, al cărui nume este dat în FCB și nu se afla anterior în tabela de fișiere. Această funcție scrie pe disc eticheta de fișier, cu lista de blocuri ocupate conținînd zerouri; deci fișie-

rul există pe disc, dar cu o lungime nulă, pînă nu se scrie ceva în el. Programatorul trebuie să verifice că nu mai există un alt fișier cu același nume, deoarece sistemul nu verifică. Dacă se poate crea noul fișier, registrul A conține un cod director cu valoarea cuprinsă între 0 și 3ı Altfel registrul A conține valoarea 0FFH (dacă, de exemplu, nu ma. există loc în tabela de fișiere a discului selectat prin octetul 0 din FCB).

23 = *Schimbă nume fișier* (Rename File).

Această funcție operează numai asupra tabelei de fișiere, modificînd numele de fișier dat în primii 16 octeți din FCB prin înlocuirea sa cu numele dat în următorii 16 octeți din același FCB. Octetul 16 din FCB, adică codul discului pentru noul nume este considerat implicit 0. Registrul A conține fie un număr între 0 și 3, fie 0FFH în caz că numele dat în primii 16 octeți din FCB nu există în tabela de fișiere a discului selectat.

30 = *Stabilire atribute fișier* (Set File Attributes)

Atributele unui fișier sînt determinate de valorile biților celor mai semnificativi din octeții 9 și 10 (tip fișier) din FCB: bitul 7 din octetul 9 este 1, dacă fișierul are atributul R/O, iar bitul 7 din octetul 10 este 1, dacă fișierul este de tipul SYS. Blocul de control fișier cu adresa în registrele DE conține atributele dorite de programator (biții respectivi pe 0 sau pe 1), iar sistemul înscrie aceste atribute în directorul discului. Eventual se pot utiliza și biții superiori din ceilalți octeți ai numelui și extensiei (se recomandă octeții 1—4 din nume) pentru alte atribute stabilite și tratate de utilizatori.

31 = *Obține adresă bloc parametri disc* (Get Disk Param. Addr.)

Se încarcă în registrele HL adresa blocului de parametrii disc din BIOS. În general această funcție nu este folosită de programele obișnuite, deoarece necesită cunoașterea structurii acestui bloc de parametri.

7.2.2. Funcții BDOS la nivel de înregistrare

Aceste funcții se folosesc pentru citirea sau scrierea de date într-un fișier deschis, corespunzător modului de exploatare.

Accesul la fișier se face numai la nivel de înregistrare de 128 octeți (egală cu un sector pe discurile flexibile de simplă densitate), indiferent de tipul discului suport și deci de dimensiunea sectoarelor fizice. În continuare se va folosi atît termenul de „înregistrare”, cît și termenul de „sector” pentru a desemna o înregistrare (un sector logic) de 128 octeți.

După executarea unei funcții BDOS de acces la un fișier în registrul A se află un cod de eroare care este zero dacă operația s-a terminat normal și are o valoare nenulă dacă s-a produs o eroare sau o condiție specială, cum ar fi încercare de citire secvențială după ultima înregistrare din fișier.

Sistemul CP/M permite două moduri de acces la înregistrările unui fișier:

— acces secvențial, numai la înregistrarea imediat următoare;

— acces direct (aleatoriu), la oricare înregistrare din fișier, folosind adresa de înregistrare în cadrul fișierului (prima înregistrare are adresa 0).

20 = *Citire secvențială* (Read Sequential)

Se citește sectorul următor din fișierul desemnat de FCB, și care trebuie să fie deschis în prealabil. Citirea se face la adresa DMA curentă, stabilită prin funcția 26. Adresa sectorului următor din fișier se află în octetul 32 din FCB, octet care este automat incrementat de sistem după citire (dar care nu este inițializat automat la deschiderea fișierului și de aceea trebuie pus 0 de către programator). Trecerea de la o zonă (de 16 koct) la alta a fișierului se face automat, inclusiv reinițializarea octetului 32 din FCB la fiecare zonă nouă. Registrul A conține 0, dacă citirea s-a terminat corect și o valoare nenulă, dacă nu s-a terminat normal (de obicei pentru că s-a citit după sfîrșitul fișierului).

21 = *Scriere secvențială* (Write Sequential)

Se scrie sectorul următor în fișierul desemnat de FCB, deschis în exploatare (funcția 15) sau în creare (funcția 22). Datele sînt luate din memorie, de la adresa DMA, stabilită în prealabil. După scriere se incrementează automat adresa de sector din octetul 32 al FCB și, eventual, se trece la următoarea zonă de 16 koct a fișierului cu reinițializarea cu 0 a octetului 32. Registrul A va conține 0 după o scriere corectă sau o valoare nenulă în caz că scrierea nu poate avea loc (de obicei, din cauză că nu mai este loc pe discul respectiv). Prin adăugarea unei noi înregistrări la sfîrșitul fișierului se realizează extinderea sa automată, în limitele spațiului disponibil pe disc.

26 = *Stabilire adresă DMA* (Set DMA Address)

Această funcție transmite sistemului, prin registrele DE, adresa zonei tampon din memorie, folosită pentru operațiile de citire (scriere) pe disc (lungimea zonei este întotdeauna de 128 octeți). La (re)inițializarea sistemului se stabilește automat adresa zonei tampon la 80H.

33 = *Citire în acces direct* (Read Random)

Se citește din fișierul desemnat de FCB sectorul a cărui adresă în fișier este dată de octeții 33 și 34 din FCB (octetul inferior în octetul 33 și cel superior în octetul 34). Octetul 35 trebuie să fie 0 (o valoare nenulă indică depășirea sfîrșitului de fișier). Citirea se face la adresa DMA curentă. După citire nu se mărește automat adresa de sector, așa cum se întîmplă la citirea secvențială, dar se trece automat la o nouă zonă a fișierului dacă este necesar. Adresele de sector în fișier pot fi între 0 și 65535. După citire registrul A conține 0 pentru citire normală sau unul dintre următoarele coduri de eroare:

01 — citire sector nescris;

03 — nu se poate închide zona curentă;

04 — căutarea unei zone nescrise (adresă de sector prea mare);
06 — adresă ce depășește dimensiunea maximă a unui fișier (octetul 35 nenul).

34 = *Scriere în acces direct* (Write Random)

Se scrie în fișierul desemnat de FCB sectorul a cărui adresă în fișier este dată de octeții 33 și 34 din FCB, de la adresa DMA curentă. Dacă este necesar, se va extinde fișierul cu un nou sector sau chiar cu o nouă zonă de 16 koct. Adresa de sector nu este modificată după scriere. Registrul A conține după scriere fie 0, dacă operația s-a terminat normal, fie un cod de eroare. Codurile de eroare sînt aceleași de la funcția 33, plus codul 05:

05 — nu mai este loc în tabelul de fișiere pentru crearea unei noi zone de fișier (unui nou segment de etichetă).

35 = *Calcul lungime fișier* (Compute File Size)

Această funcție încarcă în octeții 33 și 34 din FCB lungimea fișierului cu numele dat în FCB. Dacă fișierul conține numărul maxim de sectoare (65536), atunci octetul 35 devine 1. Trebuie observat că numărul efectiv de sectoare ce conțin date poate fi mai mic decît adresa furnizată de această funcție, dacă există goluri în fișier (dacă fișierul a fost creat în acces direct). Este posibil în acest fel să se aloce spațiu disc pentru un fișier scriind date doar în ultimul sector. Această funcție se poate utiliza și pentru extinderea unui fișier prin adăugarea de noi articole la sfîrșitul său.

36 = *Stabilire adresă de acces direct* (Set Random Record)

Se transferă adresa ultimului sector citit sau scris în mod secvențial (octetul 32) în adresa de sector folosită în acces direct (octeții 33, 34) din FCB. Această funcție se folosește pentru trecerea de la acces secvențial la acces direct sau pentru determinarea adresei unui articol căutat după conținut în mod secvențial.

Subsistemul de fișiere din CP/M (BDOS) asigură doar operațiile de bază pentru acces secvențial și direct pe bază de adresă la înregistrările fizice ale unui fișier. În alte sisteme de operare mai complexe și în unele limbaje evoluate din CP/M sînt prevăzute și alte operații de nivel superior, destinate aplicațiilor în care se lucrează cu fișiere de date:

— citire (scriere) secvențială pentru articole logice, de lungime oarecare diferită de 128 de octeți;

— citire (scriere) secvențială de octeți individuali dintr-un fișier;

— poziționare în fișier pe un anumit octet sau pe un articol cu număr dat;

— posibilitatea de adăugare de articole sau de octeți la sfîrșitul unui fișier;

— alte organizări de fișiere pentru acces direct la articole, pe bază de cheie (pe baza conținutului) ș.a.

Implementarea unor asemenea facilităţi în limbaj de asamblare sub CP/M se poate face printr-un pachet de subrutine, eventual însoţit de o bibliotecă de macroinstrucţiuni pentru apelarea lor, dar care necesită şi extinderea blocului descriptor de fişier din CP/M cu noi informaţii despre fişier (lungime articole, adresă zonă articol, număr de articole ş.a.).

Unele limbaje de nivel înalt (dBASE, COBOL, C) oferă posibilităţi de acces la fişiere de date mult extinse faţă de cele oferite prin funcţiile BDOS în limbaj de asamblare; alte limbaje de nivel înalt (FORTRAN, BASIC, PASCAL, C) asigură conversiile din format extern în format intern, necesare citirii şi scrierii de numere întregi şi reale din (în) fişiere disc.

7.2.3. Exemple de utilizare a funcţiilor BDOS cu fişiere

1) Exemplu pentru funcţiile 15, 16, 20: Fragment dintr-un program de listare la consolă a conţinutului unui fişier text, al cărui nume se dă în linia de comandă:

```
; Program de listare fisier la consola
 FCB      EQU  5CH          ;Adresa FCB implicit
 LIST:    LXI  SP,STIVA          ;zona 80H-100H folosita ca buffer
; deschidere fisier
          LXI  D,FCB         ;Deschide fisier
          MVI  C,15
          CALL 5
          CPI  OFFH         ;Verifica daca s-a deschis corect
          JZ   ER15         ;Salt daca fisier inexistent
; ciclu de citire-afisare inregistrare
 NXTREC:  LXI  D,FCB        ;Ciclu de citire/afisare sector
          MVI  C,20
          CALL 5            ;citeste inregistrarea urmatoare
          ORA  A            ;Citire corecta ?
          JNZ  EOF          ;Salt daca sfirsit de fisier
          CALL DISP         ;Afiseaza 128 octeti de la adresa 80H
          JMP  NXTREC       ;Repeta citire-afisare
; inchidere fisier
EOF:      LXI  D,FCB        ;Inchide fisier
          MVI  C,16
          CALL 5
          CALL EXIT         ;Terminare program, iesire in CP/M
ER15:     LXI  D,MES        ;Scrie mesaj de eroare la deschidere
          CALL PRINT
          CALL EXIT         ;Terminare program

 MES:     DB   13,10,'FISIER INEXISTENT','$'
          DS   64           ;Stiva proprie
 STIVA:
          END  LIST         ;adresa de lansare in executie
```

2) Exemplu pentru funcţia 26: Programul de afişare anterior rescris cu subrutine de I/E şi folosind o zonă tampon rezervată explicit:

```
  FCB       EQU  5CH         ;Bloc implicit de descriere fisier
; deschidere fisier
            LXI  D,FCB
            CALL OPEN
; stabilire adresa buffer de citire inregistrare
            LXI  D,DBUF
            MVI  C,26
            CALL 5
; ciclu de citire-afisare
  NXTREC:   LXI  D,FCB
            CALL RDSEQ        ;Citeste urmatoarea inregistrare
            JNZ  EOF          ;Subrutina RDSEQ pozitioneaza si pe Z
            CALL DISPL        ;Afisare la consola din zona DBUF
            JMP  NXTREC
; inchidere fisier
 EOF:       LXI  D,FCB
            CALL CLOSE
            ...
```

3) Exemplu pentru funcţiile 21, 22: Fragment dintr-un program de creare a unui fişier text cu date citite de la consolă (numele fişierului se dă în linia de comandă).

```
  FCB       EQU  5CH
            LXI  SP,STIVA
; deschidere fisier
            LXI  D,FCB
            MVI  C,22
            CALL 5
            INR  A            ;255+1 =0
            JZ   ER22         ;salt daca eroare la creare fisier
; ciclu de citire-scriere inregistrare
  NXTREC:   CALL GETTXT       ;citire text la adresa 80H
            JZ   EOF          ;salt daca sfirsit date la consola
            LXI  D,FCB        ;scrie o inregistrare
            MVI  C,21
            CALL 5
            ORA  A
            JNZ  ER21         ;salt daca eroare la scriere
            JMP  NXTREC
; inchidere fisier
  EOF:      LXI  D,FCB
            MVI  C,16
            CALL 5
```

4) Exemplu de utilizare reacroinstrucţiuni pentru funcţii BDOS cu fişiere. Program de copiere a unui fişier, apelat prin comanda COPY FSURSA FDEST

```
;program de copiere a unui fisier disc
FCBI EQU  5CH        ;FCB fisier sursa
#include BDOSLIB
;
CPY: LXI  SP,STK     ;stiva proprie
     MOVE FCBI+16,FCB2,16     ;muta nume fisier destinatie
     XRA  A          ;zero
     STA  FCB2+32    ;primul sector
     OPEN FCBI       ;deschide fisier sursa
     CPI  OFFH
     JZ   ER1        ;salt daca eroare la fisier sursa
     MAKE FCB2       ;deschide fisier destinatie
     CPI  OFFH
     JZ   ER2        ;salt daca eroare la fisier destinatie
SECT: GET FCBI
     CPI  O
     JNZ  STP        ;salt daca sfirsit de fisier
     PUT  FCB2
     JMP  SECT
STP: CLOSE  FCB2
     CLOSE  FCBI
     EXIT
ER1: PRINT  MES1
     EXIT
ER2: PRINT  MES2
     EXIT
;zona de date
FCB2: DS  36
MES1: DB  CR,LF,'NU EXISTA FISIERUL $'
MES2: DB  CR,LF,'DISC PLIN $'
     DS   20   ;stiva programului
STK: DS   O
     END  CPY
```

Macroinstrucţiunea MOVE transferă un şir de octeţi de la o adresă sursă la o adresă destinaţie pe o lungime dată.

5) Exemplu pentru funcţiile 33, 34: Program de scriere şi de citire fişier în acces direct, lansat prin comanda: RAND FISIER

```
; Scriere / Citire fisier in acces direct
     MACLIB  BDOSLIB          ;biblioteca de macroinstructiuni
RF   EQU  5CH  ;adresa RFILE implicit
     ORG  100H
RAND:SDMA  BUFF      ;Adresa buffer
     OPEN RF         ;Deschide fisier
     CPI  OFFH       ;Exista fisierul ?
     JNZ  DIALOG     ;Salt daca exista
     MAKE RF         ;Creare fisier nou daca nu exista
     CPI  OFFH       ;Mai este loc pe disc?
     JZ   EROARE
```

```
DIALOG:
     PRINT OPER        ;Scrie "cod operatie"
     CONIN             ;Citeste cod operatie
     CPI   'Q'
     JZ    STOP
     CPI   'R'
     JZ    READR       ;Citire de pe disc.
     CPI   'W'
     JZ    WRITER      ;Scriere pe disc
     JMP   DIALOG      ;Cere alta comanda
;Citire de pe disc si afisare la consola
READR:
     CALL RECNO        ;Citeste si pune in RFILE nr. inreg.
     CALL CLEAR        ;Sterge buffer
     READ RF           ;Citeste o inregistrare
     PRINT CRLF
     PRINT BUFF        ;Afisare inregistrare
     JMP   DIALOG      ;Citeste alta comanda
;Citire de la consola si scriere pe disc
WRITER:
     CALL RECNO        ;Citeste si pune in RFILE nr. inreg
     CALL CLEAR        ;Sterge buffer
     PRINT DATE        ;Cere introducere date
     GETCON BUFF       ;Citeste o linie dela consola
     WRITE  RF         ;Scrie pe disc
     JMP   DIALOG      ;Citeste alta comanda
;Subrutina de citire numar sector
RECNO:
     PRINT NRSEC       ;Scrie "Numar sector"
     CONIN             ;Citeste numar sector
     SUI   '0'         ;Conversie in binar numar sector
     STA   RF+33       ;Pune nr. sector in FCB
     XRA   A           ;Si un zero in oct. 34
     STA   RF+34       ;Octet superior din nr. sector
     STA   RF+35       ;octet depasire fisier
     RET
     ...
STOP: CLOSE RF   ;Inchidere fisier
     EXIT
EROARE:
     PRINT ERR
     EXIT
; Date

CRLF:   DB 13,10,'$'
DATE:   DB 13,10,'Date :$'
OPER:   DB 13,10,'Cod operatie (R,W,Q):$'
NRSEC:  DB 13,10,'Nr.Sector=$'
ERR:    DB 13,10,'Nu e loc$'
BUFF:   DS  128    ;Zona da citire/scriere
        DB   '$'
     END  RAND
```

6) Exemplu pentru funcția 19: Fragment dintr-un program care creează un nou fișier (cu nume dat în linia de comandă), dar înainte de creare șterge un eventual fișier cu același nume.

```
FCB      EQU   5CH
; sterge orice fisier anterior cu acelasi nume
         LXI   D,FCB
         MVI   C,19
         CALL  5          ;nu intereseaza rezultatul functiei 19
; creare fisier nou
         LXI   D,FCB
         MVI   C,22
         CALL  5
```

7.3. Funcții BIOS de intrare-ieșire

Funcțiile de I/E la nivel fizic, realizate de BIOS, sînt utilizate de către funcțiile BDOS, dar pot fi utilizate și direct de către programatori. Mai precis este vorba despre operațiile de I/E cu discul la nivel de pistă și de sector, care pot fi necesare în următoarele situații:

— citirea sau scrierea în pistele sistem de pe un disc CP/M (pistele 0,1);

— citirea sau scrierea în directorul unui disc CP/M (pista 2, sectoarele 1—16);

— citirea sau scrierea sector cu sector a unui disc CP/M sau non-CP/M (testarea unui disc, copierea fizică a unui disc, conversii de format logic de la un sistem de operare la alt sistem etc.).

Modulul BIOS constă din 17 subrutine, iar punctele de intrare în aceste subrutine sînt grupate la început într-un tabel de 17 instrucțiuni de salt (JMP). Acest tabel de salturi constituie interfața componentei BIOS cu componenta BDOS, dar poate fi utilizată și în unele programe de aplicații.

Adresa de început a acestui tabel este în general variabilă, depinzînd de poziția în memorie a nucleului CP/M.

Deoarece la adresa 0 se găsește întotdeauna o instrucțiune

JMP WBOOT

inseamnă că la adresele 1 și 2 se află adresa BIOS + 3, care poate fi folosită pentru a scrie programe cu funcții BIOS de I/E, dar valabile pentru orice mașină CP/M (independente de sistem). Asociind fiecărei funcții BIOS un număr (între 0 și 16) care este poziția în vectorul de salturi, atunci apelarea unei funcții BIOS se poate face astfel:

```
MVI A, cod ; cod funcție BIOS
CALL BIO
```

Subrutina BIO calculează adresa în tabel și face saltul către rutina ce realizează funcția solicitată:

```
BIO: DCR A    ;poziția față de WBOOT
     MOV B,A
     RLC      ;înmulțire cu doi
     ADD B    ;înmulțire cu 3
     MOV E, A ;adresa relativă în registrul DE
     MVI D, 0
     LHLD 1   ;adresa instrucțiunii JMP WBOOT în HL
     DAD D    ;adună adresă relativă
     PCHL     ;salt la adresa conținută în HL
```

De fapt, secvența de apelare a unei funcții BIOS este în general mai lungă, pentru că multe funcții necesită un parametru transmis prin registrul C (dacă este un octet) sau prin perechea de registre BC (dacă are doi octeți).

7.3.1. Rutine BIOS de intrare-ieșire

Funcția BOOT (0) realizează inițializarea sistemului.

Funcția WBOOT (1) realizează reinițializarea sistemului.

Funcțiile BOOT și WBOOT dau controlul monitorului de consolă CCP și nu readuc controlul în programul apelant, așa cum fac celelalte funcții BIOS.

Funcția CONST (2) înapoiază în registrul A un cod de stare al consolei cu următoarele valori posibile:

00H — dacă nu s-a introdus un caracter la consolă;

0FFH — dacă s-a introdus un caracter la consolă

Funcția CONIN (3) citește în registrul A caracterul introdus de la consolă, eliminînd bitul de paritate; dacă este necesar, se așteaptă pînă cînd se introduce un caracter.

Funcția CONOUT (4) afișează la consolă caracterul primit în registrul C (și eventual interpretează caractere de control în mod ecran). Funcțiile BIOS de citire (scriere) la consolă corespund funcției 6 din BDOS, de citire (scriere) directă la consolă.

Funcția LIST (5) afișează la dispozitivul logic LST caracterul primit în registrul C și corespunde funcției BDOS cu codul 5.

Funcția PUNCH (6) trimite la dispozitivul logic PUN, de ieșire perforator, caracterul primit în registrul C și corespunde funcției BDOS cu codul 4.

Funcția READER (7) citește în registrul A un caracter de la dispozitivul de citire (RDR); corespunde funcției BDOS cu codul 3

La utilizarea funcțiilor de scriere (ieșire) trebuie să se țină seama de diferențele în utilizarea registrelor față de funcțiile BDOS corespunzătoare: caracterul se transmite prin registrul C și nu prin registrul E.

Funcţia HOME (8) are ca efect poziţionarea capului de citire (scriere) de la unitatea de discuri selectată anterior pe pista 0; ea se reduce la funcţia SETTRK cu parametrul 0 în registrul C.

Funcţia SELDSK (9) selectează una dintre unităţile de discuri cuplate la sistem, folosind numărul primit în registrul C (0, 1, 2, 3).

Funcţia SETTRK (10) stabileşte adresa pistei în care se va citi sau scrie cu funcţiile READ/WRITE; numărul pistei (între 0 şi 76) se primeşte în registrul C.

Funcţia SETSEC (11) stabileşte adresa sectorului ce se va citi sau scrie cu funcţiile READ/WRITE; numărul sectorului (între 1 şi 26) se primeşte în registrul C.

Funcţia SETDMA (12) stabileşte adresa zonei tampon din memorie, folosită în comenzile de citire (scriere) READ/WRITE.

Adresa zonei de I/E se primeşte prin registrele BC, iar lungimea zonei se consideră implicit a fi de 128 octeţi.

Funcţia READ (13) citeşte la adresa selectată prin SETDMA un sector disc cu adresa stabilită prin SETTRK şi SETSEC de la unitatea selectată prin SELDSK.

Funcţia WRITE (14) scrie 128 octeţi de la adresa selectată prin SETDMA în pista selectată prin SETTRK şi sectorul selectat prin SETSEC la unitatea de disc selectată prin SELDSK.

Funcţia LISTST (15) depune în registrul A un octet ce indică starea dispozitivului de listare (LST), cu aceeaşi convenţie ca pentru starea consolei: A = 00H, dacă dispozitivul nu este gata şi A = 0FFH, dacă dispozitivul este gata să primească un caracter.

Funcţia SECTRAN (16) realizează renumerotarea (translatarea) numărului de sector primit în registrul C, conform tabelului de translatare cu adresa dată în registrele DE. Subrutina SECTRAN primeşte în registrul C un număr de sector logic între 0 şi 25 şi dă ca rezultat în registrul L un număr de sector fizic între 1 şi 26.

Modulul BIOS utilizează pentru operaţiile de citire (scriere) pe disc un bloc de I/E ce conţine parametrii necesari realizării operaţiei: codul operaţiei (citire (scriere)), numărul unităţii de disc, adresa pistei, adresa sectorului în cadrul pistei şi adresa de memorie a zonei de I/E. Funcţiile 9, 10, 11, 12, 16 completează pe rînd acest bloc de I/E, iar funcţiile 13, 14 introduc codul operaţiei în blocul de I/E, lansează operaţia descrisă de blocul de I/E şi aşteaptă terminarea operaţiei. În plus, la terminarea operaţiei se depune în registrul A au octet de stare care arată modul de terminare a operaţiei: A = 00H pentru terminarea normală a operaţiei şi A nenul pentru terminare anormală.

7.3.2. Exemple de utilizare a funcţiilor BIOS de I/E

Program de citire a tabelului de fişiere a discului din unitatea implicită, cu predare control în DDT:

```
;Citire director disc 0 in memorie si salt in DDT
        ORG     100H
DIR:    MVI     C,0         ;selectie disc 0
        CALL    SELDSK
        MVI     C,2         ;stabilire nr. pista
        CALL    SETTRK
        LXI     H,SECT      ;nr sector citit
        MVI     M,1
        LXI     H,DIRBUF    ;adresa de citire in memorie
        SHLD    ABUF        ;ABUF memoreaza adresa
RSEC:   LXI     H,SECT      ;stabilire nr sector citit
        MOV     C,M         ;nr sector logic in C
        CALL    SECTRAN     ;transforma in sector fizic
        CALL    SETSEC      ;stabilire nr.sector
        LHLD    ABUF        ;adresa de citire
        MOV     B,H         ;in BC
        MOV     C,L
        CALL    SETDMA      ;stabilire adresa de citire
        CALL    READ        ;citeste un sector
        LHLD    ABUF        ;calculeaza adresa urmatoare
        LXI     B,128
        DAD     B
        SHLD    ABUF        ;memoreaza adresa de citire
        LXI     H,SECT      ;calc. adresa sector urmator
        INR     M
        MVI     A,17        ;se citeasc numai 16 sectoare
        CMP     M           ;mai sint sectoare de citit ?
        JNZ     RSEC        ;repeta citirea
        RST     7           ;salt in DDT


; Subrutine pentru functii BIOS de I/E
SELDSK:
        LXI     D,8*3       ;selectie disc
        JMP     BIO
SETTRK:
        LXI     D,9*3       ;stabilire nr pista
        JMP     BIO
SETSEC:
        LXI     D,10*3      ;stabilire nr sector
        JMP     BIO
SETDMA:
        LXI     D,11*3      ;stabilire adresa de memorie
        JMP     BIO
READ:
        LXI     D,12*3      ;citire sector
        JMP     BIO
```

```
SECTRAN:
      LXI   H,XLT      ;trece sect logic in sect fizic
      MVI   B,0        ;C=nr sector logic
      DAD   B          ;HL=adresa sector fizic in XLT
      MOV   C,M        ;C=nr sector fizic
      RET
BIO:  LHLD  1          ;adresa BIOS+3
      DAD   D          ;adresa rutinei BIOS in tabela
      PCHL             ;apel BIOS
;Zone de date
DIRBUF: DS  16*128 ;zona de citire director
ABUF:   DS  2      ;adresa zonei de citire
SECT:   DS  1      ;numar sector logic

; Tabela de translatie sectoare logice-sectoare fizice
XLT: DB 1,7,13,19,25
     DB 5,11,17,23
     DB 3,9,15,21
     DB 2,8,14,20,26
     DB 6,12,18,24
     DB 4,10,16,22
     END   DIR
```

7.4. Utilizarea directă a instrucţiunilor de I/E

Dispozitivele periferice de I/E sînt cuplate la microprocesor prin intermediul unor circuite de interfaţă, care controlează şi comandă dispozitivele respective.

Unele circuite de interfaţă au un caracter mai general, permiţînd cuplarea de dispozitive diverse (de exemplu, SIO, PIO, 8255), iar alte circuite sînt specializate pentru anumite tipuri de periferice (discuri flexibile, tuburi de afişare, linii de comunicaţii etc.); o a treia categorie de circuite de I/E nu este destinată direct unor dispozitive periferice, dar este cerută de alte circuite de I/E (de exemplu, 8253, 8257, 8259 ş.a.).

Toate circuitele de interfaţă se leagă la microprocesor prin intermediul a două sau mai multe porturi de intrare-ieşire, iar instrucţiunile de I/E asigură transferul unor octeţi de date sau de control între registrul A şi porturile de I/E respective (la Z80 se poate face şi transfer între memorie şi porturile de I/E). În general cele două porturi au următoarea utilizare:

— un *port de date*, prin care se citesc sau se transmit octeţi de date;

— un *port de control şi stare*, prin care microprocesorul transmite comenzi sau citeşte starea interfeţei şi perifericului.

Unele controloare integrate reunesc mai multe circuite de acelaşi tip, dar care au o parte comună şi de aceea folosesc mai multe porturi: circuitul de ceas, canalul de acces direct la memorie ş.a.

Programarea operațiilor de I/E cu instrucțiuni ale procesorului depinde mult de circuitul de interfață utilizat și prezintă diferențe mari între sistemele cu 8080 și cele cu Z80.

Majoritatea circuitelor de interfață pot funcționa cu sau fără întreruperi, dar anumite circuite (8253, 8272, 8275, PIO ș.a.) necesită folosirea sistemului de întreruperi.

Utilizatorii microcalculatoarelor cu disc nu trebuie să programeze niciodată la nivel de porturi dispozitivele de I/E standard pentru sistemul CP/M: discul, tastatura, consola de afișare și imprimanta, de pentru care există rutine de I/E în BIOS.

O serie de aplicații necesită folosirea și programarea circuitelor de ceas (8253/CTC), de interfață paralelă (8255 /PIO) și de interfață în serie (8251/SIO); de aceea se va exemplifica utilizarea directă a instrucțiunilor de I/E și a întreruperilor de I/E cu programarea acestor circuite.

7.4.1. Utilizarea instrucțiunilor de intrare-ieșire

Microprocesorul 8080 are numai două instrucțiuni de I/E;

```
IN  p ;citește în A de la portul p
OUT p ;scrie din A la portul p
```

Alte două instrucțiuni 8080 permit controlul sistemului de întreruperi, folosit în legătură cu operațiile de I/E:

```
EI ;activare sistem de întreruperi
DI ;dezactivare sistem de întreruperi
```

Între un circuit de interfață și microprocesor are loc un dialog sub forma unui schimb de octeți prin două sau mai multe porturi succesive de I/E:

— *octeți de comandă*, de la procesor către circuit;

— *octeți de stare*, de la circuit către procesor;

— *octeți de date*, de la procesor către circuit, pentru operațiile de ieșire (de scriere), sau de la circuit către procesor, pentru operațiile de intrare (de citire).

O secvență tipică de dialog între microprocesor și un circuit de interfață arată astfel:

— se citește un octet de stare și se testează starea controlorului de I/E; dacă este nevoie se așteaptă ca interfața să fie disponibilă;

— se trimit unul sau mai mulți octeți de comandă, reprezentînd codul operației și eventual parametrii necesari operației;

— se citește sau se scrie un octet de date.

Circuitele de interfață Z80 pot primi o secvență de octeți de comandă fără testarea stării înainte de fiecare octet, dar alte circuite (8272) trebuie testate dacă sînt gata să primească un nou octet de comandă.

Octetul de stare arată dacă dispozitivul de I/E este gata să primească un octet de date sau dacă are un octet de date pe care vrea să îl trimită la microprocesor.

Atunci cînd se lucrează cu întreruperi, starea de gata a controlorului de I/E este semnalată printr-o întrerupere şi nu mai este necesar ca procesorul să testeze în buclă starea acestuia.

În general faza de execuţie a unei operaţii de I/E poate dura un timp echivalent cu mai multe sute sau mii de instrucţiuni de prelucrare internă, fie datorită modului de lucru al dispozitivului periferic (la un disc), fie datorită aşteptării unei intervenţii manuale (la tastatura locală sau la un terminal aflat la distanţă şi conectat pe legătura serială).

Această diferenţă mare de timp între operaţiile de I/E şi operaţiile din memorie a condus la funcţionarea dispozitivelor de I/E în întreruperi, ceea ce înseamnă că procesorul nu trebuie să testeze continuu terminarea unei operaţii de I/E şi că poate continua cu execuţia altor instrucţiuni, urmînd ca la terminarea operaţiei de I/E să se producă întreruperea programului în execuţie pentru tratarea sfîrşitului operaţiei.

Lucrul cu dispozitivele de I/E în întreruperi este absolut necesar pentru sistemele de operare de timp-real (cu taskuri paralele) sau cu multiprogramare şi multiacces, dar nu este necesar pentru sistemele de dezvoltare în monoacces, cum este şi sistemul CP/M.

Iată un *exemplu* de rutină de citire a unui caracter de la o tastatură, fără întreruperi:

```
KBDAT     EQU  18H          ;port de date
KBSTS     EQU  KBDAT+1      ;port de stare
CONINP:   IN   KBSTS        ;citeşte octet de stare
          ORA  A            ;testează starea
          JZ   CONINP       ;aşteaptă tastarea unui caracter
          IN   KBDAT        ;citire caracter tastat
          RET               ;ieşire cu caracter în A
```

Atunci cînd se lucrează în întreruperi, trebuie să existe pentsu fiecare dispozitiv de I/E cîte o rutină de tratare a întreruperii de I/E, rutină care are sarcina de preluare a octetului de date (la citire) sau de memorare a stării de disponibilitate şi eventual de lansare a unei noi operaţii (la scriere).

În caz că tastarea unui caracter la claviatură produce automat o întrerupere a programului în curs, rutina de tratare a acestei întreruperi arată astfel:

```
KBINT:  PUSH  PSW      ;salvare registre folosite de subrutină
        MVI   A,1      ;memorează întrerupere
        STA   KBSTAT   ;în octetul de stare KBSTAT
        IN    KBDAT    ;citeşte caracter tastat
        STA   KBCHAR   ;memorează caracter citit
        CALL  EOI      ;secvenţa de achitare a întreruperii
        POP   PSW      ;refacere A şi indicatori
        EI             ;reactivare întreruperi
        RET
```

Rutina lansată prin întreruperea de la tastatură memorează evenimentul de tastare a unei clape, precum și codul caracterului introdus, după care reactivează întreruperile și revine în programul întrerupt.

Rutina BIOS de citire a unui caracter de la consolă în varianta cu întreruperi arată astfel:

```
CONINP: LDA  KBSTAT   ;testează octet de stare
        ORA  A        ;este zero?
        JZ   CONINP   ;așteaptă tastare clapă
        XRA  A        ;zero în A
        STA  KBSTAT   ;anulează octet de stare
        LDA  KBCHAR   ;încarcă în A caracterul citit
        RET
```

Dacă se foloseau doi octeți succesivi din memorie pentru stare și pentru caracterul citit, se putea reduce lungimea celor două rutine prezentate.

Din punct de vedere al timpului de așteptare nu există practic nici o diferență între rutina de citire fără întreruperi și rutina de citire cu întreruperi, în sensul că nu se revine din rutina CONINP pînă cînd nu se apasă o clapă și deci programul care apelează această rutină nu poate face altceva între caractere succesive, oricît de mare ar fi acest interval de timp.

De fapt există o soluție pentru ca un program CP/M să nu aștepte introducerea de caractere de la consolă; această soluție este posibilă datorită existenței rutinei de testare a stării consolei, utilă în aplicațiile de timp real.

Rutina BIOS de citire stare consolă încarcă în registrul A octetul de stare asociat consolei:

```
CONSTS: LDA KBSTAT
        RET
```

În raport cu secvența generală de lucru cu un dispozitiv de I/E, citirea de la consolă se deosebește prin aceea că nu este necesar un octet de comandă, deoarece la tastatură nu este posibilă decît o singură operație. Dispozitivele de I/E care pot executa mai multe operații au nevoie de un octet de comandă care să specifice operația dorită.

Majoritatea circuitelor de interfață necesită și o fază de inițializare, în care procesorul trimite unul sau mai mulți octeți de comandă pentru stabilirea modului de lucru și alegerea parametrilor programabili.

În concluzie, programarea directă a operațiilor de I/E include în general următoarele secvențe de instrucțiuni neadiacente în memorie:

— secvența de inițializare și de stabilire a modului de lucru;

— subrutina de citire sau de scriere a unui octet;

— subrutina de tratare a întreruperilor, dacă se lucrează în regim de întreruperi.

Pentru dispozitivele cu debit mare de octeți (disc, bandă, ecran) transferul de date se face pe blocuri (sectoare, înregistrări, pagini de ecran) și nu octet, cu octet folosindu-se canale de acces direct la memorie.

Programarea cu întreruperi la circuitele din familia 8080.

Funcționarea unui dispozitiv de I/E cu întreruperi necesită prezența circuitului controlor de întreruperi 8259 precum și programarea acestui circuit.

Circuitul 8259 utilizează două porturi de I/E:

— portul p se leagă la registrul de control și stare 8259;

— portul p+1 se leagă la registrul de mascare a întreruperilor.

Cele 8 niveluri de întreruperi prevăzute de un circuit 8259 sînt mascabile individual, în sensul că întreruperile de pe fiecare nivel pot fi activate sau inhibate printr-un bit egal cu 0 (nivel activ) sau cu 1 (nivel mascat, inactiv), într-o mască de întreruperi pe un octet.

Fiecărui nivel activ trebuie să îi corespundă o rutină de tratare a întreruperilor de pe nivelul respectiv.

Rutinele de întreruperi pot fi plasate oriunde în memorie (în CP/M ele fac parte din BIOS), iar adresele lor sînt centralizate într-un tabel de salturi către aceste rutine. Tabelul de salturi are de obicei 32 de octeți, cîte 4 octeți pentru fiecare nivel, începînd cu nivelul 0, cel prioritar.

Inițializarea circuitului 8259 constă în transmiterea adresei tabelului de salturi, a lungimii acestui tabel și a numărului de circuite 8259 cuplate; inițializarea se face prin doi octeți de comandă (numiți ICW1 și ICW2) trimiși la porturile p și p+1.

Secvența de inițializare include de obicei și selectarea nivelurilor active, prin trimiterea octetului de mascare a întreruperilor, deși masca poate fi stabilită și modificată în orice moment după inițializarea circuitului. Exemplu de secvență de inițializare 8259:

```
INTCS   EQU  4OH         ; port de control (stare)
INTMK   EQU  INTCS+1     ; port mască de întreruperi
MASK    EQU  59H         ; mască pentru nivelele 0, 3, 4, 6
I8259:  DI               ; dezactivare generală a întreruperilor
        LXI  H,INTTBL    ; adresă tabelă de întreruperi
        MOV  A,L         ; octetul inferior din adresa
        ORI  16H         ; un singur 8259, interval de 4 octeți
        OUT  INTCS       ; trimite ICW1 la primul port
        MOV  A,H         ; octet superior din adresă
        OUT  INTMK       ; trimite ICW2 la al doilea port
        MVI  A,MASK      ; octetul de mascare niveluri nefolosite
        OUT  INTMK       ; trimite OCW1 la portul mască
        EI               ; activare generală întreruperi
```

Octetul ICW1 conține numai biții 5,6,7 din adresă (ultimii 5 biți din adresă sint considerați implicit zero), iar biții 0—4 sînt biți de control pentru selectarea unor opțiuni de lucru.

Iată și un exemplu de tabel de salturi (numit și vector de întreruperi) pentru un terminal TPD:

```
INTTBL: DB  0,0,0,0   ; nivelul 0: nefolosit
        JMP SIOINT    ; nivelul 1: USART 8251
        NOP
        JMP CRTINT    ; nivelul 2: ecran consolă 8275
        NOP
        DB  0,0,0,0   ; nivelul 3: nefolosit
        DB  0,0,0,0   ; nivelul 4: nefolosit
        JMP KBINT     ; nivelul 5: tastatura
        NOP
        DB  0,0,0,0   ; nivelul 6: nefolosit
        JMP FDINT     ; nivelul 7: disc flexibil
        NOP
```

După inițializare, circuitul 8259 poate primi trei octeți de comandă cu funcții separate, numiți OCW1, OCW2, OCW3. Octetul OCW1 stabilește sau modifică masca de selectare a nivelurilor active; fiecare bit din OCW1 corespunde nivelului de întrerupere corespunzător.

Octetul de comandă OCW2 trebuie transmis la terminarea tratării unei întreruperi, deoarece la producerea unei întreruperi se dezactivează automat nivelul respectiv și toate nivelurile inferioare. OCW2 reactivează aceste niveluri și poate eventual modifica prioritățile asociate fiecărui nivel. Valoarea uzuală a lui OCW2 este 20H și se transmite la sfîrșitul tratării complete sau parțiale a unei întreruperi. În cazul cel mai simplu, toate rutinele de tratare a întreruperilor se termină cu secvența următoare:

```
EOI: MVI A,20H  ; OCW2
     OUT INTCS  ; trimite OCW2 la portul de control
     POP PSW    ; PSW se salvează în stivă la început
     EI         ; reactivare generală întreruperi
     RET        ;
```

Octetul de comandă OCW3 se folosește mai rar, fie pentru a stabili un mod special de mascare, fie pentru generarea de întreruperi prin program, fie pentru citirea unor registre ale circuitului 8259.

Uneori este necesară adăugarea unor noi rutine de tratare a întreruperilor pentru dispozitive nestandard CP/M, fără să se afecteze funcționarea în întreruperi a dispozitivelor standard. În acest scop este necesară cunoașterea adresei tabelului de salturi (sau a unei copii a lui din memoria RAM) și a nivelului de întrerupere pe care este legat (dacă este legat) dispozitivul programat.

Programele care folosesc asemenea dispozitive de I/E nestandard funcționînd în întreruperi trebuie să includă:

— o secvență inițială de modificare a vectorului de întreruperi, prin introducerea unei instrucțiuni de salt la rutina de întrerupere scrisă de utilizator și de modificare a măștii de întrerupere;

— o rutină de tratare a întreruperilor de la dispozitivul respectiv;

— rutina de transfer (de citire sau de scriere);

— o secvență finală de refacere a măștii de întreruperi, pentru dezactivarea întreruperilor de la dispozitiv, înainte de terminarea programului.

Rutina de întrerupere împreună cu rutina de citire sau scriere poartă uneori numele de *„driver"* de I/E, deoarece asigură controlul unui dispozitiv de I/E și are o structură de program specifică.

O rutină de tratare a unei întreruperi de I/E sau de la ceas trebuie să includă următoarele:

— o secvență inițială de salvare în stivă a registrelor folosite de subrutină (cel puțin registrul A și indicatorii);

— o secvență de tratare minimală sau completă a evenimentului semnalat de întrerupere, pentru luarea în considerare a întreruperii;

— o secvență finală de refacere a registrelor salvate și de reactivare a întreruperilor.

Deoarece pe durata unei rutine de întrerupere sînt inactivate întreruperile de pe același nivel și de pe nivelurile inferioare, este de dorit ca executarea acestei rutine să dureze cît mai puțin și deci ca rutina să conțină cît mai puține instrucțiuni. În caz contrar, la un interval mai mic între două întreruperi succesive, este posibil să se piardă unele semnale de întrerupere și deci o funcționare eronată a programului.

7.4.2. Programarea unor circuite de interfață Intel Programarea circuitului de ceas 8253

Circuitul de ceas 8253 folosește 4 porturi succesive de I/E: primele 3 porturi sînt asociate în ordine numărătoarelor 0, 1 și 2, iar ultimul port este un port de comandă comun.

Inițializarea circuitului 8253 se face separat pentru fiecare numărător, prin trimiterea a doi sau trei octeți succesivi cu semnificația următoare:

— primul octet selectează contorul și modul de lucru pentru contorul respectiv;

— al doilea octet reprezintă partea inferioară a valorii care se încarcă în contor;

— al treilea octet reprezintă partea superioară a valorii care se încarcă în contor.

Constanta încărcată într-un contor determină perioada impulsurilor, deoarece un impuls (o întrerupere) este generată de 8253 la ajungerea în zero a contorului care numără descrescător la fiecare impuls de tact al sistemului (în modul de lucru folosit pentru funcția de ceas).

Este posibilă numai încărcarea octetului superior sau numai încărcarea octetului inferior al constantei de timp, dacă primul octet specifică acest mod de lucru.

Exemplu de secvență de inițializare circuit 8253:

```
; porturi utilizate de 8253

    CNT0   EQU  20H      ; contor 0
    CNT1   EQU  CNT0+1; contor 1
    CNT2   EQU  CNT0+2; contor 2
    CLKCS  EQU  CNT0+3; port de control/stare
; inițializare 8253

I8253:  MVI A,0B6H    ; contor 2, mod 2
        OUT CLKCS     ; selecție contor și mod
        MVI A,200     ; octet inferior pentru 20 ms
        OUT CNT2
        MVI A,0       ; octet superior constantă de timp
        OUT CNT2
```

Cuplarea rutinei de tratare a întreruperilor la nivelul 4 pe care este legat ceasul și demascarea acestui nivel se face prin secvența următoare:

```
    TMSLOT EQU   0FA50H        ; adresa în tabelul de întreru-
                                 peri 8259
           DI
           MVI   A,(JMP)       ; cod instrucțiune de salt
           STA   TMSLOT        ; în tabelul lui 8259
           LXI   H,CLKINT      ; adresa rutinei de întrerupere
           SHLD  TMSLOT+1
           IN    INTMK         ; citește masca de întreruperi
           ANI   OEFH          ; forțează bitul 4 pe 0
           OUT   INTMK         ; trimite masca nouă la 8259
           EI
```

Rutina de tratare a întreruperilor de la ceas se poate scrie astfel:

```
CLKINT:  PUSH     PSW   ; salvare registre utilizate
         PUSH     H
           LXI  H,NIMP  ; număr de impulsuri de la ceas
           INR  M       ; adună 1 la NIMP
           POP  H       ; refacere HL
           MVI  A,20H   ; reactivare întreruperi (OCW2)
           OUT  INTCS
           POP  PSW     ; refacere A
           EI
           CALL CLOCK   ; alte prelucrări
           RET
```

Rutina CLOCK actualizează ora curentă, formată din oră, minut, secundă și are un număr mai mare de instrucțiuni.

Înainte de terminarea programului care lucrează cu ceasul trebuie dezactivate întreruperile de la ceas prin secvența următoare:

```
DI
IN   INTMK  ; citeşte masca de întreruperi
ORI  10H    ; forțează pe 1 bitul 4
OUT  INTMK  ; scrie masca modificată în 8259
EI
```

Programarea circuitului de interfață paralelă 8255

Interfața paralelă 8255 se leagă pe patru porturi succesive de I/E și asigură la ieșire trei porturi a cîte 8 linii, notate cu A, B și C. Primele 3 porturi ale microprocesorului corespund porturilor A, B, C, iar ultimul port este utilizat pentru controlul interfeței 8255.

Prin trimiterea unui octet de mod la portul de control se pot configura porturile A, B, C în trei moduri diferite. Portul C poate fi utilizat în două feluri: ca port de date separat de 8 biți sau împărțit în două linii de control de cîte 4 biți asociate porturilor A (octetul superior) și B (octetul inferior).

Fiecare dintre porturile A și B, precum și fiecare dintre cele două jumătăți ale portului C pot fi programate pentru citire sau pentru scriere.

În modul 0 cele trei porturi A, B, C, se folosesc direct într-o instrucțiune IN sau OUT, fără nici un protocol de transfer.

În modul 1 (cu semnal de strob) numai porturile A și B se pot folosi ca porturi unidirecționale de date, în timp ce liniile portului C se folosesc pentru semnale de control și stare asociate porturilor A și B (cîte 4 linii pentru fiecare port).

În modul 2 portul A se poate utiliza ca port bidirecțional de date, avînd asociate 5 linii de control și stare din registrul C, iar portul B se poate utiliza ca port de date unidirecțional. Celelalte 3 linii de date din portul C pot fi utilizate fie ca linii de comandă asociate portului B, fie ca linii de date.

În exemplul de utilizare a interfeței 8255 pentru cuplarea unei imprimante paralele DZM la un terminal TPD se folosește modul 0 cu portul A ca port de ieșire și portul B ca port de intrare; transferul de date se face în paralel pe 8 biți fără întreruperi, iar imprimanta schimbă cu 8255 următoarele semnale:

PC7 = strob de transfer date în buffer imprimantă
PB7, PB4 = semnale de „imprimantă gata"
PB2 = semnal de confirmare a terminării operației
; porturi utilizate

```
PA  EQU  90H
PB  EQU  PA+1
PC  EQU  PA+2
PCS EQU  PA+3       ; port de control (stare)
        ; iniţializare 8255
I8255:  MVI A,83H  ; port A = ieşire date, port B = intrare
        OUT PCS    ; la portul de control 8255
        MVI A,80H  ; anulare strob
        OUT PC     ; la portul C
        ; rutina LIST de afişare caracter
LIST:   IN   PB    ; citeşte stare
        ANI  90H
        JNZ LIST   ; aşteaptă ca imprimanta să fie gata
        MOV A,C    ; caracter de imprimat
        ANI  7FH   ; fără bit de paritate
        CMA
        OUT PA     ; trimite octet de date
        XRA  A     ; semnal de strob la imprimantă
        OUT PC
LIST1:  IN   PB    ; citeşte stare
        ANI  40H
        JZ   LIST1 ; aşteaptă preluare date de imprimantă
        MVI A,80H  ; anulare strob
        OUT PC
LIST2:  IN   PB    ; citeşte stare
        ANI  40H
        JNZ LIST2  ; aşteaptă tipărire caracter
        RET
```

Programarea interfeţei seriale 8251

Circuitul USART 8251 asigură serializarea şi deserializarea biţilor din fiecare octet în vederea transmiterii şi recepţionării lor pe o linie de telecomunicaţie; de asemenea asigură inserarea şi verificarea biţilor de control necesari în modul asincron şi în modul sincron de transmisie.

Interfaţa 8251 foloseşte două porturi de I/E: primul port este un port de date bidirecţional, iar al doilea port este folosit ca registru de comandă şi de stare.

Circuitul 8251 trebuie iniţializat prin unul sau doi octeţi de comandă: un octet pentru modul asincron şi doi octeţi pentru modul sincron. După iniţializare este necesară o comandă de pornire a transmisiei.

Recepţia unui caracter sau terminarea emisiei unui caracter poate fi sesizată prin testarea registrului de stare sau printr-o întrerupere. De obicei se folosesc întreruperi numai pe recepţie. În registrul de stare există un bit pentru „transmiţător gata" şi un bit pentru „receptor gata".

Octetul de comandă pentru modul asincron specifică numărul biţilor de date dintr-un caracter, numărul biţilor de stop, tipul de control la paritate şi factorul de divizare al frecvenţei de comandă.

Valoarea frecvenţei cu care este comandat 8251 determină viteza de transmisie; obţinerea unei frecvenţe programabile se face folosind circuitul 8253.

Exemplu de programare a circuitului 8251.

```
SIOD  EQU  18H      ; port de date
SIOC  EQU  SIOD+1   ; port de control (stare)

; iniţializare 8253 pentru viteza de transmisie dorită
   ...

; iniţializare USART 8251

I8251:  XRA  A        ; pregătire comandă Reset
        OUT  SIOC
        OUT  SIOC
        OUT  SIOC
        MVI  A,40H    ; Reset USART
        OUT  SIOC
        MVI  A,0CEH; selecţie mod asincron, 8 biţi de date
        OUT  SIOC     ; 2 biţi de stop, fără paritate
        MVI  A,5      ; pornire USART
        OUT  SIOC

; rutina de emisie caracter din registrul C

COMOUT: IN   ;SIOC     ; citeşte octet de stare
        ANI  01H       ; test bit „transmiţător gata"
        JZ   COMOUT ; aşteaptă ca emiţătorul să fie gata
        MOV  A,C       ; octet de transmis
        OUT  SIOD      ; emisie octet
        RET

; rutina de recepţie caracter în registrul A

COMINP: IN   SIOC     ; citeşte octet de stare
        ANI  02H      ; test bit „receptor gata"
        JZ   COMINP; aşteaptă să sosească un octet
        IN   SIOD     ; preluare caracter
        ANI  7FH      ; elimină bit de paritate
        RET
```

8. PROGRAMARE ÎN LIMBAJUL C SUB CP/M

Limbajul C s-a născut şi s-a dezvoltat ca alternativă pentru limbajul de asamblare, drept care uneori este numit şi asamblor independent de maşină.

Limbajul C este un limbaj orientat maşină, fără să fie legat de o anumită maşină. Limbajul C nu este un limbaj de nivel înalt, orientat către probleme (către aplicaţii) aşa cum sînt aproape toate celelalte limbaje independente de calculator: FORTRAN, BASIC, COBOL, PL/I, Pascal, ADA etc.

Limbajul C împreună cu funcţiile sale de bibliotecă oferă programatorilor acces la maşină şi la facilităţile sistemului de operare mai mult decît oricare alt limbaj independent de maşină.

Programarea în limbajul C este ceva mai dificilă şi mai expusă la erori decît programarea în limbajele de nivel înalt menţionate, dar este mult mai simplă, mai sigură şi în orice caz mai puţin legată de echipament decît programarea în limbajele de asamblare.

Aşa cum se afirmă deseori, limbajul C este un limbaj pentru profesionişti, adică pentru cei care au drept profesie realizarea de produse program.

Aceste caracteristici situează limbajul C undeva între limbajele de asamblare şi limbajele de nivel înalt, orientate către aplicaţii.

Limbajul C posedă toate structurile de control şi multe structuri de date din limbajele de programare moderne, care fac din el un limbaj puternic, expresiv şi un bun mijloc de aplicare a principiilor moderne de programare: programarea structurată, programarea modulară, dezvoltarea în etape a programelor ş.a.

Faţă de Pascal, limbajul C stimulează mai mult dezvoltarea modulară, prin utilizarea unui număr mare de funcţii mici, repartizate în unul sau mai multe fişiere sursă, precum şi prin folosirea unor biblioteci de module sursă sau obiect.

Limbajul C este mai apropiat de modul de lucru al procesoarelor actuale decît oricare alt limbaj independent de maşină; el nu încearcă

să ascundă realitățile hardware cum ar fi: adresarea indirectă, manipularea adreselor de memorie ca variabile, operații disponibile prin instrucțiuni mașină, dar absente din multe limbaje de nivel înalt (incrementare și decrementare, deplasare, operații logice la nivel de bit ș.a.); aceste facilități hardware sînt puse la dispoziția programatorilor într-o formă simplă și independentă de orice procesor.

Această apropiere de mașină permite obținerea unui cod mașină eficient, dar calitatea codului generat depinde de programator în mai mare măsură decît pentru alte limbaje. Altfel spus, programatorul poate controla și influența mult mai bine codul mașină generat din programe scrise în C decît o poate face în limbajele Fortran, Pascal ș.a.

Limbajul C este deosebit de concis, datorită numărului redus de cuvinte cheie și utilizării unui număr mai mare de caractere speciale și combinații ale acestora.

Acest atribut al limbajului C este considerat de unii ca o calitate, deoarece permite exprimarea unor programe complexe, mari, cu un număr mic de caractere, deci fișiere sursă mai mici și toate avantajele care decurg de aici; alții consideră acest fapt ca o deficiență a limbajului, susținînd că programele C sînt mai criptice și mai expuse la erori sintactice, rezultate din absența sau prezența nedorită a unor caractere.

Limbajul C este un limbaj mic, relativ ușor de învățat și în orice caz mai ușor de compilat decît multe alte limbaje.

Remarcabil este și faptul că, deși nu există încă un standard pentru limbajul C, programele scrise în C sînt mult mai portabile între diferite sisteme decît programe scrise în Pascal, Basic sau Fortran, pentru care există de mult timp standarde.

Această portabilitate rezultă pe de o parte din faptul că toate operațiile de intrare-ieșire și de prelucrare a fișierelor (dependente de sistemul gazdă) sînt scoase în afara limbajului și realizate prin funcții de bibliotecă, dar și din faptul că nu mai este necesară extinderea limbajului C cu alte facilități (în mare parte adaptarea la un sistem particular se face prin funcțiile de bibliotecă).

8.1. Utilizarea sistemului de programe C Aztec

8.1.1. Sistemul de programe C Aztec

Firma Manx Software Systems livrează mai multe fișiere sub sistemul CP/M-80, fișiere necesare utilizării limbajului C:

CII.COM	compilator C pentru sisteme cu 8080 (vers. 1.05c)
CZII.COM	compilator C pentru sisteme cu Z80

AS.COM	asamblor relocabil format Manx
LN.COM	linkeditor format Manx
LIBUTIL.COM	bibliotecar format Manx
SIDSYM.COM	generare fişier tabelă de simboluri în format SID
LIBC.LIB	bibliotecă obiect 8080 în format Manx
MATH.LIB	bibliotecă matematică în format Manx
Z80LIBC.LIB	bibliotecă obiect Z80 în format Manx
LIBC.H	fişier antet pentru majoritatea programelor C
ERRNO.H	fişier antet pentru definire coduri de eroare
STDIO.H	fişier antet pentru funcţii standard de I/E
IO.H	fişier antet pentru funcţii de I/E de tip UNIX
OBJECT.H	fişier antet pentru programe care prelucrează fişiere obiect format Manx
FCNTL.H	fişier antet definire opţiuni funcţie open
OVLOADER.C	fişier sursă în C pentru încărcător segmente
OVBGN.ASM	fişier sursă ASM pentru încărcător segmente

Compilatoarele CII şi CZII genereazǎ fişiere de tip „ASM" care conţin cod simbolic 8080 şi urmează să fie asamblate cu AS. Fişierele obiect generate de AS au format binar, relocabil Manx cu extensia „O" şi sînt prelucrate de linkeditorul LN, care produce un fişier direct executabil de tip „COM".

Există şi posibilitatea de utilizare a programelor M80 (sau RMAC), L80 şi LIB80, deci a formatului binar relocabil Microsoft, dacă se foloseşte opţiunea „—M" la compilare.

Pentru utilizarea linkeditorului L80 este necesar ca şi bibliotecile obiect să fie în format Microsoft (format „REL"). Se recomandă următoarele nume pentru aceste biblioteci:

SOFTLIBC.REL	corespunde bibliotecii LIBC.LIB
SOFTMATH.REL	corespunde bibliotecii MATH.LIB
SOFTZLIB.REL	corespunde bibliotecii Z80LIBC.LIB

Pentru obţinerea acestor biblioteci trebuie pornit de la fişierele sursă în limbajul C şi în limbaj de asamblare care se asamblează cu M80 şi apoi se concatenează într-un singur fişier folosind bibliotecarul LIB80.

Funcţiile care intră în compunerea bibliotecii LIBC.LIB sau SOFTLIBC.REL sînt livrate în mai multe fişiere sursă, din care rezultă tot atîtea module obiect.

Cîteva fişiere sursă există în două variante: fişierul „ASM" conţine varianta originală, iar fişierul „UAL" conţine o variantă îmbunătăţită de utilizatori.

Crearea bibliotecilor obiect dintr-un număr relativ mare de module (fişiere) este facilitată prin existenţa unor fişiere de comenzi executabile prin comanda SUBMIT.

Fişierul LIBC.H este absolut necesar la compilarea oricărui program C. Lipsa acestui fişier produce eroarea „Loading error" la linkeditare cu L80.

Fişierul STDIO.H este necesar pentru utilizarea funcţiilor de intrare-ieşire de bibliotecă.

Fişierul OBJECT.H nu este necesar decît la compilarea unor programe care prelucrează formatul binar relocabil Manx.

Fişierul FCNTL.H nu este necesar în general, deoarece constantele simbolice definite în acest fişier se află şi în fişierul LIBC.H.

Fişierul ERRNO.H nu este necesar decît dacă într-un program se testează anumite coduri de eroare întoarse de către funcţiile de bibliotecă prin variabila globală „errno", folosind nume simbolice pentru aceste coduri de eroare (toate codurile de eroare sînt numere negative).

Fişierul MATH.H poate fi necesar atunci cînd se apelează funcţii din biblioteca matematică MATH.LIB sau SOFTMATH.REL; se poate şi fără acest fişier antet, dacă se declară cu „double" funcţiile matematice utilizate.

8.1.2. Utilizarea compilatorului C Aztec

În cazul utilizării formatului Manx secvenţa de comenzi minimală pentru executarea unui program scris în limbajul C este de forma:

```
CII TEST.C
AS TEST.ASM
LN TEST, LIBC
TEST
```

În cazul utilizării formatului Microsoft secvenţa de comenzi are forma următoare:

```
CII —M TEST.C
M80 = TEST.ASM
L80 TEST, SOFTLIBC/S, TEST/N/E
TEST
```

Dacă lipseşte numele fişierului de intrare în comanda de compilare, atunci compilatorul aşteaptă introducerea textului sursă de la consolă şi face o compilare incrementală (linie cu linie) punînd codul generat în fişierul specificat cu opţiunea „O" sau afişînd acest cod la consolă, dacă lipseşte numele fişierului de ieşire.

Este recomandabil să se utilizeze un fişier de comenzi lansat prin comanda SUBMIT pentru obţinerea programelor executabile din programe C.

De remarcat că dacă apar erori la compilare, atunci se şterge automat fişierul de comenzi în curs de execuţie („$$$.SUB") şi deci nu se mai execută asamblarea şi linkeditarea.

Fiecare din programele CII, AS, LN, LIBC pot avea o serie de opțiuni în linia de comandă, opțiuni formate dintr-o literă și precedate de caracterul „—" (minus), în sintaxa sistemului UNIX.

Cîteva opțiuni de compilare (opțiuni CII):

M generare cod sursă pentru asamblare cu M80 (implicit se generează cod pentru asamblorul AS)

Od: nume.tip fișierul de ieșire se pune pe unitatea de disc „d" sub numele „nume.tip" (implicit se generează fișierul de ieșire pe unitatea de unde s-a citit fișierul de intrare, cu numele fișierului de intrare și cu extensia „ASM")

T se introduc în fișierul de ieșire, sub formă de comentarii, liniile sursă C din care s-a generat codul respectiv

V se afișează informații suplimentare asupra codului generat

După compilare cu programul CII (sau CZII) există două posibilități de continuare a prelucrării codului generat:

— folosind produsele Manx: AS și LN

— folosind produsele Microsoft M80 și L80 (sau RMAC şi L80)

Iată cîteva elemente pentru compararea celor două variante.

— Timpul de asamblare și de linkeditare este aproximativ același pentru cele două variante, cu un ușor avantaj pentru programul LN față de L80.

— Dimensiunea bibliotecilor și fișierelor în format binar relocabil este sensibil mai mică pentru formatul Microsoft (REL) decît pentru formatul Manx (O și LIB). De exemplu, biblioteca LIBC.LIB are 44 koct, biblioteca MATH.LIB are 18 koct, în timp ce biblioteca SOFTLIBC.REL are 28 koct, iar SOFTMATH.REL are 13 koct.

Adăugînd la aceasta și faptul că fișierele AS.COM (26 koct) și LN.COM (24 koct) sînt mai mari decît fișierele M80.COM (20 koct și L80.COM (11 koct), rezultă pentru varianta Microsoft un spațiu disc mai mare disponibil pentru programele compilate și asamblate.

— Programele M80 și L80 sînt mai bine cunoscute și documentate decît programele AS și LN ca opțiuni de lucru, mesaje de eroare.

— Linkeditorul LN (Manx) prevede facilități pentru crearea unor programe segmentate.

Compilatorul C Aztec, ca și alte compilatoare, poate să afișeze o lungă listă de erori sintactice, toate însă cu o singură cauză, cum ar fi lipsa unei declarații, un delimitator în plus sau în minus etc.; de aceea este important să fie văzut și înțeles primul mesaj de eroare, celelalte fiind de multe ori consecințe ale primei erori și nu folosesc la depistarea erorii.

Afișarea mesajelor de eroare poate fi oprită temporar și reactivată apoi cu caracterul de control CTRL/S (^S), iar toată compilarea poate fi terminată forțat cu caracterul CTRL/C.

8.1.3. Segmentarea programelor mari

Un program C care conduce la un program executabil prea mare în raport cu memoria disponibilă poate fi împărțit în mai multe segmente, dintre care unele se încarcă la aceleași adrese de memorie (se suprapun în memorie).

Un program segmentat are în general o structură de arbore și constă dintr-un segment rădăcină și mai multe segmente paralele pe unul sau mai multe niveluri. La segmentele suprapuse se poate face apel din rădăcină sau dintr-un segment paralel.

Realizarea unui program segmentat cu linkeditorul Manx LN necesită prezența fișierelor OVLOADER.O și OVBGN.O, obținute din fișierele sursă OVLOADER.C și OVBGN.ASM. Funcția OVLOADER este un încărcător de fișiere format COM care permite și transmiterea unor date segmentului încărcat.

Particularități ale programelor segmentate

— Funcția program principal din segmentele diferite de rădăcină trebuie să aibă numele „ovmain", dar în rădăcină are numele „main".

— În rădăcină trebuie apelată funcția „settop" pentru stabilirea adresei de sfîrșit a întregului program segmentat care este și adresa zonei alocate dinamic pentru date prin funcția „alloc". Argumentul funcției „settop" este lungimea celei mai lungi ramuri din arborele de segmentare și se determină pe baza adresei de bază și dimensiunii fiecărui segment, afișate la linkeditare.

— Dacă se folosesc în segmente funcții de intrare/ieșire pentru fișiere cu buffer asociat (fopen, fread, fwrite, fgets, fputs, getc, putc, fseek, s.a.), atunci trebuie să existe un apel al unei asemenea funcții și în rădăcină (de exemplu, un printf) pentru ca să se includă în rădăcină zonele de date necesare acestor funcții.

— Un segment poate face apel la alt segment, chiar dacă segmentul la care se face apel se suprapune peste segmentul care face apel. La linkeditarea segmentului care face apel trebuie folosită opțiunea — R pentru generarea unui fișier RSM, folosit la linkeditarea segmentului la care se face apel; această opțiune nu este necesară pentru ultimul segment de pe fiecare ramură.

— Toate segmentele diferite de rădăcină trebuie să includă modulul OVBGN.O.

— Comunicarea de date între segmente se poate face fie prin variabile externe, fie prin argumente transmise la încărcare prin intermediul funcției OVLOADER.

Pentru segmentarea programelor C se va proceda astfel:

— fiecare segment de program se constituie într-un fișier sursă separat, compilat și asamblat într-un fișier obiect separat;

— se linkeditează segmentul rădăcină și celelalte segmente neterminale cu opțiunea — R și împreună cu fișierul OVLOADER.O; opțiunea — R generează un fișier de tip RSM, necesar la legarea segmentelor terminale;

— se linkeditează segmentele terminale folosind fișierul RSM al segmentului care apelează și încarcă segmentul respectiv, precum și fișierul OVBGN.O; linkeditorul LN creează din fișierul obiect rădăcină un fișier executabil de tip COM, iar din fișierele segmentelor fișiere executabile de tip OVR;

— se lansează în execuție prin comandă operator fișierul rădăcină.

Exemplu. Secvența de comenzi necesară obținerii fișierelor executabile pentru un program cu un segment rădăcină ROOT și cu două segmente OVL1 și OVL2 suprapuse la aceleași adrese (codul în ultimul exemplu din paragraful 8.4).

```
CII ROOT.C
AS ROOT.ASM
CII OVL1.C
AS OVL1.ASM
CII OVL2.C
AS OVL2.ASM
LN — R ROOT.O OVLOADER.O LIBC.LIB
LN OVL1.O OVBGN.O ROOT.RSM LIBC.LIB
LN OVL2.O OVBGN.O ROOT.RSM LIBC.LIB
```

8.2. Particularități ale limbajului C Aztec

8.2.1. Prezentare sumară a limbajului C

Elemente lexicale ale limbajului C

Unitățile lexicale ale limbajului C sînt: identificatori, cuvinte cheie, constante, șiruri, operatori și separatori.

O categorie aparte de separatori o formează „spațiile albe", care cuprind următoarele caractere: blanc (0x20 în ASCII), tab (0x09), linie nouă (LF = 0x0A) și comentariile.

În afară de locurile unde este necesar un spațiu alb pentru separarea unor identificatori, cuvinte cheie sau constante succesive, aceste spații albe sînt ignorate de compilator și pot fi utilizate oriunde în program.

Comentariile sînt delimitate prin perechile de caractere:

/* ca început de comentariu

*/ ca sfîrșit de comentariu

Un identificator este un nume simbolic, format din litere și cifre și care începe cu o literă. Caracterul „—" („underscore" = subliniere) este considerat ca literă. Numai primele 8 caractere dintr-un identifica-

tor sînt semnificative, dar un identificator mai lung de 8 caractere nu este considerat ca o eroare.

În C literele mici sînt diferite de literele mari, deci „int" este altceva decît „INT".

Numele simbolice se folosesc fie pentru identificarea unor zone de date (ca nume de variabile) fie pentru identificarea unor funcţii (ca nume de funcţii).

Cuvintele cheie ale limbajului C sînt cuvinte rezervate, care nu pot fi utilizate drept identificatori. Ele se scriu cu litere mici. Cuvintele cheie ale limbajului C sînt:

auto, *break*, *case*, *char*, *continue*, *default*, *do*, *double*, *else*, *extern*, *float*, *for*, *goto*, *if*, *int*, *long*, *register*, *return*, *short*, *sizeof*, *static*, *struct*, *switch*, *typedef*, *union*, *unsigned*, *while*.

În C există următoarele tipuri de *constante*:

întregi, caractere, reale

Constantele întregi pot avea 3 lungimi diferite:

întregi scurţi *(short)*,

întregi normali *(int)*,

întregi lungi *(long)*

Constantele întregi pot fi exprimate în următoarele sisteme de numeraţie:

zecimal;

octal (dacă încep cu cifra 0);

hexazecimal (dacă încep cu Ox sau OX).

Literele A,...,F folosite ca cifre hexazecimale pot fi scrise fie ca litere mari, fie ca litere mici: a, b, c, d, e, f.

În multe implementări constantele scurte nu există sau se confundă cu constantele normale. Constantele lungi, atunci cînd există, sînt fie constante întregi mai mari decît valoarea maximă a întregilor normali, fie constante întregi terminate cu litera L sau cu litera mică 'l', în oricare dintre cele trei baze.

O constantă de tipul caracter este formată dintr-un caracter încadrat între ghilimele simple (de exemplu 'x'); valoarea unei asemenea constante este egală cu codul numeric al caracterului respectiv în codificarea utilizată de fiecare calculator.

Pentru caracterele de control uzuale se folosesc următoarele secvenţe de caractere afişabile:

\b pentru BS (spaţiu înapoi, cod ASCII 0x08)
\n pentru LF (linie nouă, cod ASCII 0x0A)
\r pentru CR (cap de linie, cod ASCII 0x0D)
\f pentru FF (pagină nouă, cod ASCII 0x0C)
\t pentru TAB (tabulator, cod ASCII 0x09)
\\ pentru '\' (caracter afişabil şi nu de control)
\; pentru apostrof (')

\0 pentru NUL (cod ASCII 0x00)
\ddd pentru codul octal al caracterului (d = 0..7)

Dacă caracterul special '\' este urmat de un alt caracter decît cele arătate, atunci el este ignorat de compilator.

Constantele reale pot avea o parte întreagă, un punct zecimal, o parte fracționară, o literă „E" sau „e" și un exponent precedat eventual de un semn (+ sau —). Pot lipsi fie partea întreagă, fie partea fracționară, dar nu ambele. Poate lipsi punctul zecimal sau litera „E" și exponentul, dar nu ambele simultan. Toate constantele reale sînt considerate implicit în dublă precizie.

Un șir ("string") este o succesiune de caractere încadrată de ghilimele duble ("). Compilatorul C adaugă automat un octet nul (\0) la sfîrșitul fiecărui șir, pentru a permite recunoașterea sfîrșitului șirului.

În consecință, constanta 'a' este diferită de șirul "a" prin aceea că șirul are doi octeți; atunci cînd apar ca argumente la apelarea funcțiilor diferența este și mai mare, deoarece în cazul lui "a" se transmite valoarea caracterului, iar în cazul lui "a" se transmite adresa șirului (adresa caracterului 'a').

Un șir are tipul „tablou de caractere" și clasa „static".

Toate șirurile dintr-un program sînt memorate distinct, chiar dacă sînt mai multe șiruri identice.

Într-un șir se pot utiliza toate caracterele afișabile și orice caracter neafișabil precedat de '\'; pentru a include într-un șir caracterul apostrof dublu se folosește secvența \".

Nume, valori și adrese în limbajul C

În limbajul C un nume de variabilă poate reprezenta fie o valoare (numerică sau nenumerică) din memorie, fie o adresă de memorie. Spre deosebire de alte limbaje (Fortran, Pascal ș.a.), în C se pot manipula atît valori cît și adrese, în cadrul expresiilor, deci practic în toate instrucțiunile.

O variabilă de tip adresă *(pointer)* este o variabilă ce conține adresa unei alte variabile, a unui tablou sau a unei structuri.

Existența unor variabile de tip adresă (pointer) permite două moduri de referire la date din memorie:

— adresarea directă, printr-un identificator, a valorii datelor;
— adresarea indirectă, prin intermediul unei variabile pointer și a operatorului de indirectare '*'.

De exemplu, valoarea unei variabile *x* se poate obține fie folosind identificatorul *x* (adresare directă), fie folosind expresia *p (adresare indirectă), dacă variabila pointer p conține adresa variabilei *x*.

Operatorul de adresare '&' aplicat unui nume de variabilă dă ca rezultat adresa variabilei respective. Exemplu:

p = &x;

Operatorul de indirectare, '*' se folosește și în declararea variabilelor de tip pointer. De exemplu:

int x, *p;

declară variabila *x* ca fiind de tip întreg și variabila *p* ca fiind o adresă de întreg (un pointer către un întreg).

Variabilele de tip pointer și operatorul de adresare indirectă nu se folosesc însă de obicei pentru referirea la date simple, care au un nume (întregi, reali, caractere); ele se folosesc fie pentru referirea datelor de orice tip alocate dinamic (cu funcția alloc) și care nu au un nume, fie pentru adresarea datelor structurate: tablouri, structuri, uniuni. Exemple:

&a sau &a[0] reprezintă adresa tabloului „a", sau mai corect adresa primului element din tabloul „a".

a[i] și *(a + i) sînt echivalente și desemnează elementul i din tabloul a.

În limbajul C un identificator de tip tablou, structură sau uniune nu este un nume pentru întreg tabloul (structura, uniunea) ci este interpretat ca adresă a primului element din tablou sau din structură; această interpretare are loc în expresii și în argumentele funcțiilor. Exemplu:

int n, a[50], amax;
amax = max (n, a); sau amax = max (n, &a); sau amax = max(n, &a[0]);

Deoarece o valoare din memorie poate fi referită în C fie printr-un nume, fie printr-o indirectare, s-a introdus noțiunea de „*referință*", care include toate modalitățile de referire la date din memorie (în documentația originală de C se folosește termenul „lvalue", prescurtare de la „left value" adică parte stîngă, deoarece o referință de variabilă este tot ceea ce poate apărea în partea stîngă a unei atribuiri de valoare).

Un identificator în limbajul C are două atribute:
— clasa de memorare;
— tipul valorilor pe care le poate lua identificatorul.

Există 4 clase de memorare ce pot fi declarate explicit:
— automat („auto")
— static („static");
— extern („extern");
— registru („register")

Variabilele din clasa „*auto*" sînt alocate dinamic, la execuție, în stivă la intrarea în blocul unde sînt definite și dispar din memorie (se scot din stivă) la ieșirea din blocul respectiv.

Variabilele din clasa „*static*" sînt alocate static, la compilare, și ocupă memorie pe toată durata execuției programului.

Domeniul de valabilitate a variabilelor automate și statice este limitat la blocul în care sînt definite, deci un nume de variabilă are aceeași semnificație numai în cadrul unui bloc.

Variabilele *externe* sînt alocate static și au ca domeniu de valabilitate întreg programul; ele sînt variabile globale, utilizate pentru comunicarea între funcții compilate împreună sau separat.

Variabilele din clasa *„register"* sînt tot variabile „auto", dar care sînt asociate, dacă este posibil, unor registre ale mașinii și nu unor locații de memorie.

În limbajul C sînt prevăzute trei tipuri de date de bază:

— tipul *caracter* („char");
— tipul *întreg* („int");
— tipul *real* („float").

Cu aceste tipuri simple se pot construi alte tipuri structurate (sau tipuri derivate) prin:

— *funcții* cu rezultat de orice tip;
— *tablouri* („arrays") cu componente de același tip;
— *adrese* ale datelor de orice tip („pointers");
— *structuri* cu componente de tipuri diferite („structures");
— *uniuni* de componente diferite („unions").

În general aceste metode de construire a noi tipuri se pot aplica succesiv și recursiv, deci putem avea funcții cu argumente de tipul adresă sau tablouri de structuri sau tablouri de tablouri și structuri cu componente structuri sau uniuni sau tablouri.

Tipul întreg poate avea 4 variante:

— întregi cu semn („int");
— întregi fără semn („unsigned");
— întregi scurți („short");
— întregi lungi („long").

Lungimea fiecărei variante de întreg depinde de implementarea limbajului pe fiecare mașină, fiind posibil să existe numai una sau două lungimi de întregi (tipul „unsigned" există întotdeauna).

Cuvintele cheie „short", „long", „unsigned" și „register" pot fi folosite singure sau să preceadă ca adjective cuvîntul „int".

Tipul real are două variante:

— reali în precizie simplă („float");
— reali în precizie dublă („double").

În funcție de implementare poate exista o singură precizie la tipul real.

Relațiile între diferite tipuri de date pot fi rezumate astfel:

— Un caracter sau un întreg scurt se poate utiliza oricînd în locul unui întreg.

— Transformarea unui caracter într-un întreg se poate face cu sau fără extinderea semnului, funcție de implementare; în cazul folosirii codului ASCII pe 7 biți, caracterele se transformă în întregi pozitivi.

— Transformarea unui întreg mai scurt într-un întreg mai lung se face cu extinderea semnului la stînga.

— Transformarea unui întreg fără semn într-un întreg lung se face prin completare la stînga cu zerouri.

— Conversia unui întreg lung într-un întreg scurt sau într-un caracter se face prin trunchiere la stînga, reținîndu-se partea din dreapta.

— Toate operațiile între reali se fac în precizia maximă, dacă există două feluri de reali.

— Adresele (pointerii) sînt tratate ca întregi fără semn, dar rezultatul scăderii a două adrese este un număr cu semn.

La stabilirea tipului rezultatului unei expresii aritmetice acționează în ordinea prezentată următoarele reguli:

— operanzii de tip „char" și „short" sînt transformați în tipul „int", iar operanzii de tip „float" sînt transformați în „double";

— dacă un operand este de tip „double", atunci toți ceilalți operanzi sînt convertiți în „double", iar rezultatul este tot de tipul „double";

— dacă un operand este de tipul „long", atunci toți ceilalți sînt aduși la tipul „long", iar rezultatul este de tip „long";

— dacă un operand este de tip „unsigned" toți ceilalți sînt aduși la acest tip, iar rezultatul este de tip „unsigned";

— dacă nu există operanzi de tipurile „double", „long", „unsigned" atunci rezultatul este de tip „int".

Expresii în limbajul C

În limbajul C expresiile pot fi deosebit de complexe, datorită numărului mare de operatori existenți, printre care și operatori de atribuire. Ca urmare evaluarea unor subexpresii dintr-o expresie poate avea efecte laterale („side effects"), cu consecințe asupra rezultatului final al expresiei.

Ordinea de aplicare a operatorilor depinde de prioritatea asociată fiecărui operator și de prezența parantezelor.

Tratarea excepțiilor aritmetice depinde de implementare, dar în general depășirile la operațiile cu întregi nu sînt semnalate.

Expresiile sînt formate din operanzi și operatori.

Operanzii pot desemna valori sau adrese de memorie, iar rezultatul unei expresii poate fi o valoare sau o adresă.

Vom denumi operand de tip referință sau referință („lvalue") un operand care poate apărea în partea stîngă a unei atribuiri.

Noțiunea de referință este importantă, deoarece anumiți operatori necesită operanzi de tip referință, iar alți operatori produc rezultat de tip referință.

Un *operand referință* poate avea următoarele forme:

— identificator de variabilă simplă;

— element de tablou (variabilă indexată);

— element de structură, desemnat prin numele structurii;

— element de structură, desemnat prin adresa structurii;
— indirectare printr-un pointer.

Este posibil și uneori necesar ca oricare dintre aceste forme să fie încadrate de paranteze rotunde; parantezele rotunde utilizate la o referință nu modifică interpretarea acesteia, deci (referință) este echivalent cu referință.

Operanzii care pot apărea în expresii constituie cea mai simplă formă de expresie, numită expresie primară. O *expresie primară* este fie o referință, fie una dintre următoarele:

— constante de orice fel;
— șiruri de caractere încadrate de ghilimele duble ('');
— nume de funcții cu sau fără argumente, dar urmate oricum de parantezele rotunde, specifice funcțiilor.

Un identificator declarat ca tablou este de tip pointer și are asociată adresa primului element din tablou. În mod similar, un șir (șir de caractere între ghilimele duble) are ca valoare adresa primului caracter din șir deoarece un șir este asimilat cu un tablou de caractere. Aceste observații sînt importante mai ales în cazul utilizării șirurilor și tablourilor ca argumente la apelarea unor funcții.

Notația referință [expresie] desemnează un element dintr-un tablou, deci o variabilă indexată. Exemple:

x[i], x[0], x[2*i + 1], a[i] [j], a[2] [3], *a[k]

Notația referință (listă-expresii) sau referință ( ) reprezintă apeluri de funcții cu și fără argumente.

Tipul rezultatului unei funcții rezultă din definiția funcției sau, în lipsa unei declarații explicite de tip, tipul este „int''.

În C transmiterea argumentelor la funcții se face numai *prin valoare*, deci prin copierea valorilor din funcția care face apel într-o stivă de argumente folosită de funcția la care se face apel.

Argumentele de tip „char'' și „short'' sînt convertite automat în tipul „int'', iar argumentele de tip „float'' sînt trecute automat la tipul „double''. Numele de tablouri sînt automat înlocuite cu adresele tablourilor respective.

Deoarece argumentele funcțiilor pot fi adrese (pointeri), iar funcția poate modifica datele de la aceste adrese, efectele unei funcții nu se rezumă în general la rezultatul asociat numelui funcției (ceea ce face inutilă noțiunea de subrutină).

Ordinea de evaluare a expresiilor ce constituie argumentele efective nu este garantată, ci depinde de fiecare implementare.

Notația referință. identificator desemnează un membru al unei structuri sau uniuni; „referință'' se referă la o structură sau la o uniune, iar „identificator'' este numele componentei. Exemplu:

```
struct { int re, im; } c1, c2, c3;
....
c3.re = c1.re + c2.re;
```

Notația referință → identificator desemnează un membru al unei structuri sau uniuni a cărei adresă este „referință" și care are numele „identificator". Exemplu:

```
struct lnode { int val ; / * valoare nod */
               struct lnode *link ; /* legătura la alt nod */
             } ;
struct lnode *p ;
. . . .
printf ("%d", p → val); /* scrie valoarea de la adresa p */
p = p → link; /* adresa nodului următor */
```

Operatorii din expresiile primare ( ) [ ] . → au prioritate maximă față de ceilalți operatori și se evaluează de la stînga la dreapta.

În general o *expresie* poate avea una dintre formele următoare:

```
expr-primară
* expresie
& expresie
— expresie
! expresie
~ expresie
+ + referință
— — referință
referință + +
referință — —
expresie op-binar expresie
referință op-atribuire expresie
expresie ? expresie : expresie
expresie, expresie
(nume-tip) expresie
sizeof expresie
sizeof (nume-tip)
```

Operatorii unari sînt:

```
* & — ! ~ + + —— sizeof (tip)
```

Operatorii binari sînt:

```
* / %
+ —
>> <<
< >   <= >=
= =   ! =
&
↑
|
&&
| |
? :
```

Operatorii de atribuire sînt:

= += −= *= /= %= >>= <<= &= ↑= |=

Operatorii unari au prioritate mai mică decît operatorii din expresiile primare, dar au prioritate mai mare decît operatorii binari.

Ordinea de acțiune a operatorilor unari este de la dreapta la stînga.

Operatorii binari și operatorul condițional (? :) au priorități descrescătoare în ordinea enumerării lor, iar operatorii de aceeași prioritate (din aceeași linie) se aplică de la stînga la dreapta.

Operatorii de atribuire au toți aceeași prioritate și acționează de la dreapta la stînga.

Operatorul virgulă (,) are cea mai mică prioritate și acționează de la stînga la dreapta.

În continuare se descrie efectul fiecărui operator.

Operatorul unar * are sensul de indirectare; el se aplică asupra unei expresii de tip pointer și are ca rezultat o referință. Deci expresia *p este înlocuită cu valoarea de la adresa conținută în variabila pointer p.

Operatorul unar & este complementar lui *; aplicat asupra unei expresii de tip referință el are ca rezultat un pointer, deci adresa valorii respective.

Operatorul unar — (semnul minus) schimbă semnul operandului pe care îl precede. Nu există operator unar pentru semnul plus.

Operatorul unar ! are sensul de negare logică; ! e este 1 dacă „e" are valoarea zero și !e este 0 dacă „e" este nenul..

Operatorul unar ~ are ca rezultat complementul față de 1 al valorii expresiei pe care o precede, deci inversare bit cu bit.

Operatorul unar ++ are sensul de incrementare, deci de adunare cu 1 a operandului.

Operatorul unar −− are sensul de decrementare, deci de scădere a lui 1 din valoarea operandului.

Operatorii ++ și −− pot apărea înainte sau după operand, efectul lor depinzînd în general de poziția față de operand.

Expresia v++ produce valoarea lui v și apoi adună 1 la această valoare; expresia ++v adună mai întîi pe 1 la valoarea lui v și produce ca rezultat valoarea incrementată.

Aplicat asupra unui identificator operatorul ++ (sau −−) are același efect indiferent de poziția lui față de operand.

Aplicat asupra unei referințe de forma *p efectul este diferit:

++*p înseamnă *p = *p + 1

*p++ înseamnă *p, p = p + 1

Incrementarea unui pointer nu înseamnă întotdeauna adunarea cu 1, ci adunarea cu o valoare egală cu lungimea tipului de date pe care îl adresează acel pointer.

Operatorul unar *(tip)* are ca efect conversia valorii operandului la tipul specificat în paranteză. Această construcție (numită „cast" de autorii limbajului) acționează ca o funcție de conversie a tipului operandului. Exemple:

(float) i
(int) x

Operatorul unar *sizeof* are ca rezultat numărul de octeți ocupați de un operand sau de un tip de date. Aplicat asupra unui nume de tablou dă numărul de octeți ocupat de tot tabloul:

sizeof tab
sizeof long

Operatorul binar * are sensul de înmulțire a două numere oarecare.

Operatorul binar / are sensul de împărțire a două numere oarecare.

Operatorul binar % are ca rezultat restul împărțirii întregi a două numere; semnul restului este același cu semnul deîmpărțitului.

Operatorul binar + este operatorul de adunare.

Operatorul binar — este operatorul de scădere.

Adunarea unui întreg la un pointer se face după înmulțirea întregului cu lungimea valorii adresate de pointer, iar rezultatul este tot un pointer. Dacă p este adresa unui element dintr-un tablou, atunci p + 1 este adresa elementului următor, indiferent de tipul tabloului (de lungimea fiecărui element).

Asupra variabilelor de tip pointer mai sînt posibile operația de scădere a unui întreg dintr-un pointer și de scădere a doi pointeri. Este posibil ca rezultatul scăderii a doi pointeri (un întreg cu semn ajustat la dimensiunea valorilor adresate) să nu aibă sens, din cauza cerințelor de aliniere la anumite adrese de memorie. Cazul uzual de folosire este cu doi pointeri în cadrul unui tablou.

La operațiile aritmetice +, —, *, /, % se realizează automat conversiile de tip menționate: din „float" în „double", din „char" în „int" precum și conversiile necesare în expresiile neomogene: din „int" în „long" și din „int" în „float".

Operatorii binari << și >> necesită operanzi întregi și realizează deplasarea la stînga («), respectiv la dreapta (>>) a primului operand cu un număr de poziții binare egal cu cel de al doilea operand. Primul operand este tratat ca o configurație binară. Deplasarea se face cu introducere de zerouri pe la o extremitate și prin pierderea cifrelor binare care ies pe la cealaltă extremitate.

Operatorii binari de relație < > <= >= au un rezultat de tipul „int", egal cu zero, dacă relația nu este adevărată, și egal cu 1, dacă relația este adevărată. Dacă sînt necesare se fac automat conversiile aritmetice menționate.

Este posibilă compararea a doi pointeri, dar rezultatul acestei operații este sigur numai dacă cei doi pointeri adresează elemente dintr-un același tablou.

Operatorul == arată relația de egalitate între operanzi.

Operatorul != arată relația de inegalitate între operanzi.

Operatorii == și != au prioritate mai mică decît ceilalți operatori de relație; rezultatul lor este 1, dacă relația este adevărată, și 0, dacă relația nu este adevărată.

Un pointer se poate compara cu un întreg, dar rezultatul este funcție de implementare. Un pointer egal cu zero nu adresează nimic (nu indică adresa 0).

Operatorul binar & realizează produsul logic bit cu bit („și" logic); exemplu: 0xAA & 0x33 = 0x22.

Operatorul binar | realizează suma logică bit cu bit („sau" logic); exemplu: 0xAA | 0x 33 = 0x BB.

Operatorul binar ↑ realizează suma logică modulo 2 („sau" exclusiv); exemplu: 0x AA ↑ 0x 33 = 0x 99.

Operatorul binar && are rezultat 1 dacă ambii operanzi au valori diferite de zero (operatorul logic „și").

Operatorul binar || are ca rezultat 1, dacă cel puțin un operand are o valoare diferită de zero (operatorul logic „sau").

Spre deosebire de operatorii & și |, operatorii && și || se evaluează întotdeauna de la stînga la dreapta, iar dacă primul operand este 0 (pentru &&) sau 1 (pentru ||), atunci nu se mai evaluează și al doilea operand.

Operatorul binar condițional ? : se folosește în expresii de forma:

```
expr1 ? expr2: expr3
```

și se interpretează astfel: dacă expr1 are o valoare diferită de zero, atunci valoarea expresiei expr2 este rezultatul; dacă expr1 este zero, atunci valoarea lui expr3 este rezultatul.

Efectul expresiei condiționale poate fi obținut astfel:

```
if (expr1) expr2;
else expr3;
```

Operatorul binar de atribuire trebuie să aibă la stînga o referință, iar la dreapta poate avea orice expresie. Rezultatul expresiei de atribuire are tipul operandului din stînga.

Efectul operatorului de atribuire simplu = este înlocuirea valorii operandului din stînga cu valoarea operandului din dreapta.

Efectul operatorilor de atribuire compuși de forma op = este echivalent cu efectul cumulat al operatorului binar „op" și al atribuirii. Expresia

```
e1 op = e2
```

este echivalentă ca efect cu expresia

```
e1 = e1 op e2
```

dar expresia e1 este evaluată o singură dată.

Pentru operatorii de atribuire += și —= operandul din stînga poate fi un pointer. Atribuirile de întregi la pointeri sau de pointeri la întregi pot conduce la erori de adresare; singura atribuire garantată este atribuirea constantei zero la un pointer, care rezultă într-un pointer nul, diferit de orice alt pointer.

Operatorul binar "," (virgula) se folosește între două expresii și are efectul evaluării în ordine a celor două expresii, iar ca rezultat valoarea celei de a doua expresii.

Atunci cînd pot apărea confuzii cu alte utilizări ale virgulei, se vor utiliza paranteze. Exemplu:

f (a, (t = 3, t + 2), b)

În anumite situații se pot folosi numai expresii constante, adică expresii care au drept rezultat o constantă. O asemenea expresie poate conține numai constante întregi, constante de tip caracter și operatorii următori:

— operatorii unari —, ~, sizeof

— operatorii binari +, —, *, /, %, &, |, ↑, <<, >>, ==, !=, <, >, <=, >=, ?:

Se pot folosi de asemenea paranteze în expresiile constante.

Instrucțiuni ale limbajului C

Instrucțiunile de bază ale limbajului C sînt următoarele:

— instrucțiunea expresie;
— instrucțiunea condițională (if);
— instrucțiunea de ciclare cu test inițial (while);
— instrucțiunea de ciclare cu inițializare (for);
— instrucțiunea de ciclare cu test final (do);
— instrucțiuni de selecție caz (switch, case, default);
— instrucțiunea de revenire din funcție (return);
— instrucțiunea de ieșire forțată (break);
— instrucțiunea de continuare ciclu (continue);
— instrucțiunea de salt (goto).

Este prevăzută și posibilitatea de a folosi instrucțiuni compuse, adică blocuri de instrucțiuni încadrate între acolade și tratate ca o singură instrucțiune.

Deși instrucțiunile de control if, while, for, do, switch, break, continue permit scrierea programelor C fără a recurge la instrucțiuni de salt, există totuși și o instrucțiune goto precum și posibilitatea de etichetare a instrucțiunilor.

Etichetele de instrucțiuni sînt identificatori, urmați de caracterul ':' și pot fi referite numai în funcția unde sînt definite.

Este prevăzută în C și o instrucțiune vidă, necesară pentru anumite situații de programare.

Cu excepția instrucțiunii compuse, orice altă instrucțiune trebuie terminată prin caracterul ';'.

Instrucţiunea compusă (numită şi bloc de instrucţiuni) grupează între acolade mai multe instrucţiuni (eventual şi declaraţii) şi poate fi utilizată oriunde poate apărea o instrucţiune. În mod uzual instrucţiunile compuse intervin la instrucţiunile if, while, do, for şi switch.

La intrarea într-un bloc care conţine declaraţii de variabile se alocă memorie pentru variabilele de tipul „auto" sau „register", declarate în acel bloc; semnificaţia acestor variabile se termină la terminarea blocului în care sînt declarate.

Instrucţiunea expresie

expresie;

este cel mai utilizat tip de instrucţiune şi reprezintă de obicei atribuiri sau apeluri de funcţii.

Instrucţiunea condiţională poate avea două forme:

if (expresie) instrucţiune

if (expresie) instrucţiune else instrucţiune

care se interpretează astfel: se evaluează expresia din paranteză şi, dacă valoarea acestei expresii este nenulă, atunci se execută instrucţiunea imediat următoare; dacă există şi cuvîntul „else", atunci instrucţiunea ce urmează lui „else" se va executa în caz că expresia testată are valoarea zero. În cazul utilizării mai multor instrucţiuni „if" incluse, regula de împerechere a lui „else" cu „if" este cea folosită şi în alte limbaje: un „else" face pereche cu ultimul „if" fără „else" întîlnit.

Instrucţiunea de ciclare cu condiţie iniţială

while (expresie) instrucţiune

produce execuţia repetată a instrucţiunii conţinute atîta timp cît valoarea expresiei din paranteză este diferită de zero. Deoarece are loc mai întîi testarea expresiei, este posibil ca instrucţiunea din ciclu să nu se execute nici o dată.

Instrucţiunea de ciclare cu condiţie finală

do instrucţiune while (expresie);

execută în mod repetat instrucţiunea conţinută pînă cînd valoarea expresiei testate devine zero. Deoarece testul se face după executarea instrucţiunii din ciclu, această instrucţiune se execută cel puţin o dată.

Instrucţiunea de ciclare cu iniţializare

for (exp1; exp2; exp3) instrucţiune

este echivalentă ca efect cu secvenţa următoare:

exp1;

while (exp2) {instrucţiune; exp3;}

Expresia „exp1" specifică iniţializările necesare ciclului, expresia „exp2" specifică testul care se face înainte de fiecare iteraţie, iar expresia „exp3" specifică operaţiile efectuate după fiecare iteraţie (de obicei incrementări sau decrementări ale unei variabile contor). Oricare dintre cele trei expresii poate lipsi; cînd lipseşte expresia de testat

„exp2", ea este considerată implicit egală cu 1, deci produce repetarea fără sfîrşit a ciclului.

Instrucţiunile de selecţie între mai multe cazuri se folosesc în construcţii de forma următoare:

```
switch (expresie)
{case expc1: instrucţiune; instrucţiune;...
case expc2: instrucţiune; instrucţiune;...
...
default: instrucţiune; instrucţiune;...
}
```

cu observaţia că nu este obligatoriu ca ultimul caz să fie cazul introdus prin „default".

Interpretarea acestei construcţii este următoarea: se evaluează expresia de selecţie din paranteză (expresie întreagă) şi se compară cu valoarea fiecărei expresii constante expc1, expc2,...; dacă valoarea unei expresii dintr-o linie prefixată prin „case" este egală cu valoarea expresiei de selecţie, atunci se execută instrucţiunile din linia respectivă şi toate care urmează; dacă valoarea expresiei de selecţie diferă de toate expresiile „case", atunci se execută instrucţiunile prefixate prin „default" (cazul implicit) sau, în lipsa lui „default" nu se execută nimic.

De reţinut diferenţa faţă de Pascal sau alte limbaje: după ce se execută instrucţiunile unui caz, se trece în secvenţă la instrucţiunile următoare, chiar dacă acestea corespund altor cazuri. Rezultă că ordinea enumerării cazurilor este în general importantă şi că mai multe cazuri pot folosi în comun un grup de instrucţiuni.

Pentru separarea cazurilor prin ieşirea forţată din blocul „switch", după executarea instrucţiunilor unui caz se foloseşte „break":

```
switch (exp)
   { case 0: instrucţiune; ...break;
     case 1: instrucţiune; ...break;
     ....
   }
```

Instrucţiunea „break"

```
break;
```

produce terminarea sau ieşirea forţată din cicluri programate cu „while", „do", „for" sau din blocuri introduse prin „switch". Dacă sînt mai multe instrucţiuni incluse de tipurile menţionate, atunci „break" termină instrucţiunea cea mai interioară.

Instrucţiunea „continue"

```
continue;
```

produce continuarea (reluarea) ciclului cel mai interior realizat cu una din instrucţiunile „while", „do", „for", fiind echivalentă cu un salt din locul unde se află „continue" la locul de continuare a ciclului, adică după toate instrucţiunile din ciclu.

Instrucţiunea „return"

return;

return expresie;

produce un salt înapoi la funcţia apelantă. Pentru prima formă rezultatul funcţiei este nedefinit (probabil o funcţie fără rezultat direct); pentru a doua formă, valoarea expresiei este valoarea asociată numelui funcţiei, deci rezultatul ei direct.

În lipsa instrucţiunii „return", se consideră implicit că există o asemenea instrucţiune imediat înaintea acoladei finale din funcţia respectivă.

Instrucţiunea de salt

goto identificator

are ca efect un salt la eticheta definită prin „identificator" din cadrul aceleaşi funcţii.

Instrucţiunea vidă

;

poate fi utilă pentru a folosi o etichetă la sfîrşitul unui bloc de instrucţiuni sau pentru un ciclu fără instrucţiuni în corpul ciclului. Exemplu:

```
while (*dest++ = *sursa++);
```

Oricare dintre tipurile de instrucţiuni menţionate pot fi precedate de o etichetă de forma

identificator:

Exemplu:

```
go to error;
...
error: putchar ('?'); return;
```

Declaraţii în limbajul C

Declaraţiile sînt necesare pentru a specifica interpretarea fiecărui identificator de către compilator, iar unele declaraţii au ca efect şi o rezervare de memorie, eventual însoţită şi de iniţializarea zonei cu valori constante.

În principal declaraţiile sînt necesare pentru variabile, deşi uneori trebuie declarate şi funcţiile.

Ca regulă generală toate variabilele trebuie declarate înainte de a fi utilizate, dar unele implementări admit ca argumentele de tip „int" să nu mai trebuiască declarate în funcţii.

Pentru funcţiile cu rezultat diferit de „int", este necesară declararea tipului funcţiei. Funcţiile al căror tip nu este explicit declarat sînt considerate de tip întreg.

Un program şi fiecare funcţie dintr-un program conţin de obicei mai multe declaraţii, în diferite locuri; poziţia unei declaraţii de variabile este în general importantă, deoarece determină domeniul de valabilitate al variabilelor declarate.

Argumentele unei funcţii trebuie declarate imediat după antetul funcţiei, dar înainte de prima acoladă cu care începe definirea funcţiei.

Exemplu:

```
toupper (c)
char c;
{char ch;
ch = c;...
}
```

O declarație poate avea în general patru părți:

— un specificator al clasei de memorare:
auto, static, extern, register, typedef;

— un specificator al tipului datelor:
  — char, short, int, long, unsigned, float, double,
  — specificator de structură sau uniune,
  — nume de tip

— o listă de obiecte declarate (numite și declaratori);

— o listă de valori pentru inițializarea variabilelor declarate.

În majoritatea cazurilor practice declarațiile conțin numai un specificator de tip și o listă de declaratori.

Atributele „short", „long", „unsigned" pot fi utilizate fie împreună cu „int", fie singure, dar cu aceeași semnificație.

În lipsa unui specificator pentru clasa de memorare se consideră următoarele convenții implicite:

— o variabilă declarată într-o funcție are clasa „auto"
— o variabilă declarată în afara oricărei funcții are clasa „extern"
— o funcție are clasa „static".

Obiectele declarate pot fi variabile valorice, pointeri, nume de tablouri sau funcții, iar declaratorii pot avea una dintre formele următoare:

— identificator;
— (declarator);
— *declarator;
— declarator ( );
— declarator [expr-const];
— declarator [ ].

Ultima formă corespunde unui tablou cu dimensiunea neprecizată fie deoarece tabloul este argument formal al unei funcții, fie deoarece dimensiunea rezultă din inițializare.

Aspectul declarației este similar modului de utilizare al variabilei sau funcției declarate, adică o variabilă sau o funcție apare în declarații la fel cum apare în expresii (urmată de paranteze rotunde sau drepte, precedată de operatorul de indirectare ș.a.m.d.).

Ca regulă generală, obiectul descris de către declarator are tipul și clasa de memorare specificate în declarație.

Exemple de declarații cu diferiți declaratori:

```
int i; /* i este o variabilă întreagă */
int *p; /* p este o variabilă de tip adresă de întreg */
```

```
int f( ); /* f este o funcție întreagă*/
int a[  ]; /* a este un tablou de întregi */
int *f( ) ; /* f este o funcție de tip adresă de întreg */
int *a [10]; /* a este un tablou de adrese de întregi */
int **a; /* a este adresa unui pointer către un întreg */
int t[10] [20]; /* t este un tablou de 10 tablouri cu cîte 20 de întregi
fiecare */
```

În declarații, ca și în expresii, parantezele pot modifica ordinea de interpretare a operatorilor *, ( ), [ ]. Exemple:

```
int (*p) (  ); /* p este un pointer către o funcție întreagă */
int (*p) [10]; /* p este un pointer către un tablou */
```

Declarații de structuri și de uniuni.

O structură („structure") este un obiect compus din mai mulți membri, în general de tipuri diferite; fiecare membru al structurii poate avea un nume, un tip și eventual o dimensiune.

Componentele unei structuri se mai numesc și cîmpuri („fields"). Un cîmp fără nume poate fi necesar pentru alinierea cîmpurilor care urmează la anumite adrese.

Specificatorul unei structuri poate avea una dintre formele:

```
struct {lista-decl-str}
struct identificator {lista-decl-str}
struct identificator
```

Ca și la celelalte declarații, acest specificator de structură este urmat în general de unul sau mai mulți declaratori, de obicei identificatori care denumesc variabile de tipul structurii specificate.

Prima formă specifică o structură anonimă. Exemplu:

```
  struct {
  float re; float im;
} c1, c2, c3 [10];
```

Variabilele c1 și c2 sînt structuri cu cîte două cîmpuri, iar c3 este un tablou de asemenea structuri (de fapt numere complexe).

A doua formă a specificatorului de structură atribuie și un nume structurii descrise. Exemplu:

```
  struct complex {
  float re, im;
  } c1, c2, c3 [10];
```

În original acest fel de nume atribuit structurii se cheamă „structure tag" și se deosebește de numele atribuit cu „typedef" prin modul în care va fi utilizat ulterior.

Este posibil ca specificatorul de structură, ca și oricare alt specificator de tip, să nu fie urmat de nici un declarator, ceea ce ar permite asocierea unui (altui) nume al tipului respectiv. Exemplu:

```
  struct complex
  {float re, im; }
```

După oricare dintre cele două declarații anterioare se poate utiliza cea de a treia formă a specificatorului de structură în care se menționează doar numele structurii definite anterior. Exemplu:

```
struct complex x, y, *px;
```

Cuvîntul cheie „typedef" permite atribuirea unui nume fie unor tipuri de bază, fie unor tipuri derivate, utilizîndu-se de obicei pentru structuri. Exemple de declarații „typedef":

1) typedef char *text;

După această declarație se poate scrie de exemplu:

```
text fintrare, fiesire, lista [50];
```

2) typedef struct {float re, im;} complex;

După această declarație se poate scrie de exemplu:

```
complex x, y, *px;
```

A se observa diferența față de cealaltă declarație pentru aceleași variabile.

O uniune („union") se declară la fel ca o structură, dar este interpretată diferit: obiectul uniune poate avea ca valoare la un moment dat numai valoarea unui singur membru al uniunii. Tipul uniune este util, atunci cînd într-o aceeași variabilă trebuie memorate valori de tipuri diferite pe parcursul unui program. Rezervarea de memorie pentru o uniune este determinată de lungimea maximă necesară pentru fiecare dintre cîmpurile sale.

Cîmpurile componente ale unei structuri sau ale unei uniuni pot fi referite în două feluri:

— direct prin numele structurii urmat de '.' și de numele cîmpului:

```
x.re    x.im
```

— indirect prin adresa structurii urmată de '→' și de numele cîmpului:

```
px→ re    px→ im
```

Sînt permise cîmpuri de biți cu sau fără nume, urmate de ':' și de lungimea cîmpului exprimată în număr de biți. Exemplu:

```
struct {
unsigned stare: 2;
unsigned priorit: 8;
                  : 5;
char *nume;
} task;
```

În definirea unei structuri sau uniuni se pot folosi alte structuri sau uniuni definite anterior sau chiar structura în curs de definire (structuri cu autoreferire). Exemplu: declararea unui nod de arbore binar

```
struc nod {
int valnod;
struct nod *stinga;
struct nod *dreapta;}
```

Orice declarație de variabilă simplă (nu tablouri sau structuri) poate fi urmată de o *inițializare* constînd din semnul de atribuire ' = ' și de o expresie constantă. Exemplu:

char semn = '+';

Nu pot fi inițializate tablourile din clasa „auto", dar pot fi inițializate tablourile din clasa „extern" sau „static" printr-o listă de inițializare.

Lista de inițializare este încadrată între acolade și poate conține la rîndul ei alte liste de inițializare; o listă de inițializare este o listă de expresii constante, separate prin virgule. Exemplu:

float t [4] [4] = { {1,1}, {2,2}, {3,3} }

Este o inițializare parțială a unei matrice și ilustrează utilitatea sublistelor de inițializare; efectul acestei inițializări este: t[1] [1] = 1, t[1] [2] = 1, t [2] [1] = 2, t [2] [2] = 2,...

Inițializarea unui tablou de caractere se poate face într-o formă simplificată folosind un șir de caractere în locul unei liste de constante de tip caracter. Exemplu:

char err[ ] = "eroare \n"

în loc de:

char err[ ] = {'e', 'r', 'o', 'a', 'r', 'e', '\n'};

Variabilele externe și statice sînt inițializate automat cu zero de către compilator, dar variabilele „auto" (din stivă) nu sînt inițializate automat și pot avea orice valori.

Structura programelor C

Un program C este compus dintr-o funcție program principal, cu numele „main", și eventual alte funcții și declarații de date externe; aceste funcții și definiții de variabile nu trebuie să se afle toate într-un singur fișier sursă, deci nu trebuie compilate toate împreună.

Compilarea separată a unor părți de program necesită ca variabile (sau funcții) definite într-un fișier, dar utilizate (și) în alte fișiere să fie declarate folosind cuvîntul „extern".

O *„definiție"* de variabilă sau de funcție este altceva decît o *„declarație"* de variabilă sau de funcție: definirea unei funcții sau variabile se face o singură dată (în unul din fișierele programului), dar pot apărea mai multe declarații pentru o singură variabilă sau funcție (în fișierele care utilizează variabila sau funcția, dar care nu conțin definiția acesteia).

Definirea unei variabile implică și alocarea de memorie pentru variabila respectivă, de aceea poate fi însoțită și de o inițializare a variabilei. Declararea unei variabile anunță doar tipul variabilei și nu alocă memorie.

Variabilele *externe* sînt variabile definite în afara oricărei funcții; definirea variabilelor externe nu necesită folosirea cuvîntului cheie „extern".

Cuvîntul „extern" este necesar numai pentru declararea unor variabile externe, definite într-un alt fișier. Prezența acestui cuvînt arată că pentru variabilele respective se alocă memoria într-un alt fișier.

Într-un program multifişier trebuie să existe un fişier şi numai unul care să conţină o definiţie pentru o variabilă externă (deci o declaraţie fără cuvîntul „extern"). Exemplu:

într-un fişier: int i; a[100];

în alt fişier: extern int i, a[ ];

Ca sintaxă, o declaraţie de variabilă arată la fel cu o definiţie de variabilă, cu următoarele deosebiri:

— cuvîntul „extern" poate apărea doar în declaraţii;

— dimensiunile unui tablou nu trebuie specificate decît la definirea sa (pentru alocare de memorie).

O declaraţie cu „extern" se poate utiliza în afara funcţiilor sau în interiorul unor funcţii.

Declararea unor variabile cu „extern" poate fi necesară şi pentru un program format dintr-un singur fişier, dacă variabila respectivă este utilizată înainte de a fi definită.

Toate funcţiile dintr-un program C sînt implicit externe.

Utilizarea într-un fişier a unei funcţii definite într-un alt fişier nu necesită declararea cu „extern" a numelui funcţiei, dar poate fi necesară declararea tipului funcţiei, dacă acest tip este diferit de „int". Ca şi în alte limbaje (FORTRAN de exemplu) şi în C compilatorul generează automat declaraţii externe pentru toate funcţiile apelate.

O declaraţie de funcţie conţine doar numele funcţiei urmat de paranteze, dar fără argumente. Exemplu:

double sin ( ), cos ( );

O definiţie de funcţie constă din numele funcţiei însoţit de argumentele formale în paranteză, urmat de declaraţii pentru aceste argumente şi de un bloc de instrucţiuni şi declaraţii între acolade, care reprezintă corpul funcţiei.

Domeniul de valabilitate al argumentelor formate ale unei funcţii se termină o dată cu funcţia respectivă.

Domeniul de valabilitate al variabilelor declarate în interiorul unei funcţii este fie funcţia în întregime, fie un bloc de instrucţiuni din cadrul funcţiei, dacă declaraţia face parte din blocul respectiv.

Domeniul de valabilitate lexical al unei variabile externe declarate în afara funcţiilor este textul sursă care urmează declaraţiei în fişierul sursă respectiv. Acesta este domeniul în care un nume simbolic de variabilă este asociat cu acelaşi obiect din memorie.

Domeniul de valabilitate al unei variabile externe se poate extinde şi la alte fişiere sursă, dacă se folosesc declaraţii cu „extern" pentru această variabilă în fişierele respective.

Prin convenţie variabilele externe, declarate explicit „static", nu sînt accesibile decît în fişierul sursă respectiv, fiind inaccesibile pentru funcţii din alte fişiere. Şi funcţiile locale pot fi declarate „static".

Din punct de vedere semantic variabilele externe se folosesc ca variabile comune sau globale, accesibile pentru mai multe funcţii ale

unui program şi reprezintă o altă modalitate de comunicare între funcţii decît comunicarea prin argumentele şi prin rezultatul direct al funcţiei.

În limbajul C este posibil ca o funcţie să se apeleze pe ea însăşi, deci sînt permise funcţii recursive.

Execuţia unui program C începe cu prima instrucţiune din funcţia „main", deci cu programul principal.

Funcţia „main" poate fi definită cu două argumente sau fără argumente, după cum programul respectiv foloseşte sau nu date transmise prin linia de comandă care apelează programul.

Limbajul C prevede în mod standard posibilitatea transmiterii de date unui program prin linia de comandă. Aceste date sînt de obicei nume de fişiere cu care lucrează programul respectiv, dar pot avea orice formă şi orice semnificaţie, deoarece fiecare program îşi interpretează cum doreşte parametrii din linia de comandă.

În C se consideră ca argument în linia de comandă orice şir de caractere delimitat de blancuri sau alte spaţii albe.

Pentru prelucrarea *argumentelor din linia de comandă* de către programele C, este prevăzut ca funcţia program principal „main" să poată avea doi parametri cu semnificaţia următoare:

— primul parametru (denumit uzual „argc") este un întreg egal cu numărul argumentelor din linia de comandă;

— al doilea argument (denumit uzual „argv") este un tablou de adrese ale şirurilor de caractere ce reprezintă valorile argumentelor din linia de comandă.

Prin convenţie, argv [0] conţine numele programului (numele comenzii), argv [1] conţine valoarea primului argument, argv [2] conţine valoarea celui de al doilea argument etc. Exemplu:

copy vechi.c. nou.c

Funcţia „main" primeşte următoarele argumente:

argc = 2, argv [0] = „copy",
argv [1] = „vechi.c", argv [2] = „nou.c"

Directive preprocesor

Orice compilator C conţine un preprocesor, care caută în textul sursă şi interpretează toate liniile care încep cu caracterul „#". Aceste linii de control al compilării se numesc directive preprocesor şi asigură următoarele facilităţi:

— substituţii textuale ale unor şiruri de caractere;
— includerea unor fişiere sursă la compilare;
— compilare condiţionată a unor porţiuni de text sursă;
— atribuirea unor numere liniilor sursă pentru depanare;

Directiva

#*include* "nume-fişier" sau

#include <nume-fişier>

este înlocuită de preprocesor prin conţinutul fişierului menţionat, care trebuie să fie un fişier sursă C. Exemplu:

#include "STDIO.H"

Directiva

#*define* identificator şir

înlocuieşte peste tot în textul sursă (cu excepţia constantelor de tip caracter sau şir de caractere) orice apariţie a identificatorului cu şirul de caractere conţinut în directivă. Şirul se termină la sfîrşitul liniei sursă, care conţine directiva define. (comentariile sînt ignorate ca de obicei).

Directiva

#define identificator (identificator,...) şir realizează atît înlocuirea primului identificator (numele macro), cît şi înlocuirea numelor din paranteze (argumentele macro) prin elemente ale şirului de substituţie. Exemplu:

#define putchar (c) putc (c, stdin)

Un apel al funcţiei putchar de forma

putchar (x)

va fi înlocuit de preprocesor prin:

putc (x, stdin)

Directiva

#*undef* identificator

anulează efectul unei directive #define, referitoare la acelaşi identificator.

Directiva

#*if* expr-const

testează dacă valoarea expresiei constante „expr-const" este diferită de zero.

Directiva

#*ifdef* identificator

testează dacă identificatorul menţionat a apărut într-o directivă # define.

Directiva

#*ifndef* identificator

testează dacă identificatorul nu este definit pentru preprocesor.

Pentru toate cele trei directive „if" liniile care urmează pînă la o directivă #endif sau #else sînt incluse în compilare, dacă condiţia testată este satisfăcută şi sînt ignorate, dacă condiţia nu este satisfăcută. Liniile dintre directivele #else şi #endif sînt compilate, dacă condiţia testată de directiva #if nu este satisfăcută.

Directiva

#*else*

introduce un bloc de linii sursă tratate de compilator, atunci cînd condiţia testată de directiva #if nu este adevărată.

Directiva

#*endif*

termină un bloc de linii sursă introdus prin directiva #if

Directiva

#*line* constantă identificator

stabilește pentru linia sursă următoare un număr de linie egal cu valoarea constantei și eventual stabilește pentru fișierul sursă numele identificatorului (dacă este folosit în directivă).

Compilatorul C însoțește orice mesaj de eroare sintactică de numele fișierului sursă compilat și de numărul liniei sursă în cadrul acestui fișier. Prin directiva #line se pot forța alte numere de linie decît cele atribuite în secvență de compilator, ceea ce poate facilita localizarea erorilor.

8.2.2. Particularități de implementare C Aztec

Față de versiunea de referință a limbajului C din cartea de prezentare (Kernighan, Ritchie) versiunea Aztec prezintă unele deosebiri:

— variabilele definite în exteriorul funcțiilor nu au o valoare inițială implicită (nu sînt inițializate cu zero);

— se pot face atribuiri între structuri;

— există cuvîntul cheie „enum", folosit pentru declararea de date prin enumerarea valorilor posibile;

— argumentele formale de funcții pot fi declarate ca fiind de clasa „register", pentru a permite o adresare mai eficientă în cadrul funcției la aceste argumente (practic, unul dintre argumente se transferă în registre procesor);

— există cuvintele cheie „asm" și „endasm", folosite pentru inserarea de instrucțiuni mașină în programe C;

— funcțiile standard de bibliotecă care însoțesc compilatorul C Aztec sînt în majoritatea lor identice cu funcțiile standard C din sistemul UNIX. Pe lîngă biblioteca de funcții C de intrare-ieșire mai există o bibliotecă matematică de funcții asemănătoare celor existente în Fortran, pentru aplicații de calcul (calcule în precizie dublă).

Precizări asupra limbajului C Aztec

— Limbajul C Aztec admite toate tipurile de date prevăzute în manualul de referință, cu următoarea reprezentare internă:

— caractere (char): 1 octet (8 biți) în cod ASCII;

— întregi și întregi scurți (int, short): 2 octeți (16 biți) valori între —32 768 și + 32 767;

— întregi fără semn (unsigned): 2 octeți (16 biți) valori între 0 și 65 535;

— întregi lungi (long): 4 octeți (32 biți) valori între —2**—31 și 2**31;

— reali dublă precizie (double): 8 octeți (64 biți)

valori între —2**—63 și 2**63 cu precizie de 16 cifre zecimale.

— sînt admise toate tipurile de constante. Exemple:

'a', '\n', '\012'	constante de tip caracter
"a", "abc"	constante de tip șir
27, —27	constante întregi zecimale
033, —033	constante întregi octale
0x1b, 0x1B	constante hexazecimale
10L, 101	constante întregi lungi
3.14, 0. 314 e1, 314E-2	constante reale

— Alocarea memoriei și declararea variabilelor cu diferite clase de memorare:

— "auto": memorie alocată dinamic în stiva adresată de SP;

— "static": memorie alocată static cu DB, DW, în bloc comun;

— "extern": memorie alocată static, declarare cu ENTRY, EXTRN;

— "register": o singură variabilă de 2 octeți în registrele BC.

— Sînt admise structuri și uniuni cu cîmpuri de orice tip, inclusiv cîmpuri de biți.

— Declararea tipului funcțiilor întregi și declararea tipului argumentelor întregi de funcții pot fi omise. Exemplu:

```
int func (a1, a2)
int a1, a2; {...}
```

se poate scrie și astfel:

```
func (a1, a2) {...}
```

— Există toate directivele preprocesor din manualul de referință, inclusiv macro cu argumente la #define.

Următoarele forme ale directivei „include" sînt echivalente:

```
#include STDIO.H
#include "STDIO.H"
#include <STDIO.H>
```

Orice program C Aztec are nevoie de o directivă pentru includerea unui fișier antet, absolut necesar.

— Fișierele standard de I/E ale limbajului C (stdin, stdout) sînt asociate consolei sistem (dispozitivului "CON:").

— Sînt prevăzute toate funcțiile standard de I/E pentru fișiere cu buffer asociat structurate în articole, fișiere referite printr-un pointer.

— Sînt disponibile de asemenea funcții de I/E, specifice sistemului UNIX, pe blocuri de octeți de orice lungime, cu fișiere referite printr-un descriptor de fișier (număr întreg).

— *Șirul format* la funcția *„scanf"* poate conține:

— specificatori de conversie, constînd din caracterul „%", eventual un caracter „*" de ignorare a valorii citite, eventual un număr pentru lungimea maximă a cîmpului și un caracter de conversie;

— spații albe (blanc, tab, linie nouă), care sînt ignorate;
— alte caractere, care trebuie să corespundă cu următorul caracter diferit de spațiu alb din șirul citit.

Specificatorii de format acceptați în „scanf" sînt:

%c pentru un singur caracter (un spațiu alb este tratat ca orice alt caracter);
%d pentru întregi scriși în zecimal;
%o pentru întregi scriși în octal (precedați sau nu de cifra 0);
%x pentru întregi în hexazecimal (cu sau fără Ox în față);
%h pentru întregi scurți (short);
%s pentru șiruri de caractere;
%f pentru numere neîntregi în orice format corect;
%e pentru numere neîntregi în orice format corect.

Formatul numerelor neîntregi introduse include: opțional un semn (+ sau —), un șir de cifre care poate conține un punct zecimal, opțional un exponent format din litera E (sau e) urmat de un întreg cu sau fără semn.

Specificatorii d, o, x, f, e pot fi precedați de litera „l" (L mic) care arată că valorile citite sînt întregi sau reali lungi.

Șirul de caractere citit prin funcția „scanf" se poate extinde pe mai multe linii (caracterul LF este ignorat) și este format în general din mai multe cîmpuri. Un cîmp de intrare este un șir de caractere diferite de spații albe și care se termină la primul spațiu alb, dacă nu se specifică lungimea cîmpului în format, sau la primul caracter separator care apare între specificatorii de format, sau după numărul de caractere indicat ca lungime a cîmpului în specificatorul de conversie.

Pentru a citi următorul caracter diferit de spațiu alb din șirul de intrare, se va folosi specificatorul %1s și nu %c.

— *Șirul format* la funcția *„printf"* poate conține două tipuri de elemente:

— specificatori de conversie precedați de caracterul %;
— alte caractere, inclusiv spații albe, copiate în șirul de ieșire.

Specificatori de format admiși în funcția „printf":

%c pentru un singur caracter;
%s pentru șiruri de caractere (se scriu atîtea caractere cîte are șirul din memorie sau cîte indică numărul de cifre de după punct);
%d numere întregi, în zecimal;
%u pentru numere întregi fără semn, în zecimal;
%o pentru numere întregi fără semn, în octal;
%x pentru numere întregi fără semn, în hexazecimal;
%f pentru numere neîntregi cu parte întreagă și parte fracționară (sub forma [—] mmm.nnnnnn la care numărul de cifre de la partea fracționară este indicat explicit sau este 6 implicit);

%e pentru numere neîntregi cu mantisă și exponent (sub forma [—]m. nnnnnnE[—]xx, la care mantisa are numărul de cifre specificat după punct sau implicit 6 cifre);

%g pentru numere neîntregi în forma cea mai scurtă.

În cadrul specificatorului de format, între % și litere care arată tipul conversiei, pot apărea în ordinea enumerării, următoarele:

— semnul minus, care indică alinierea la stînga în cîmpul extern;

— un număr întreg n, care specifică lungimea minimă a zonei în care se va afișa o valoare (la printf) sau lungimea maximă a zonei din care se citește o valoare (la scanf);

— un alt număr întreg separat prin punct de lungimea cîmpului (n.m); acest număr m specifică lungimea maximă la afișarea unui șir sau numărul de cifre la partea fracționară a numerelor neîntregi;

— litera „l" care arată că valoarea respectivă este „lungă"; se poate folosi atît pentru numere întregi, cît și pentru numere neîntregi.

Dacă lipsește lungimea cîmpului, atunci se folosesc atîtea poziții cîte sînt necesare pentru afișarea în întregime a valorii respective (nu se trunchiază niciodată numerele afișate și nici nu se refuză afișarea, dacă cîmpul specificat este prea mic).

Exemple de specificatori de conversie:

%d, %7d, %—20u, %10.4f, %101x, %.101f

Caracterul % urmat de orice alt caracter în afara celor menționate are ca efect scrierea caracterului respectiv; de exemplu %% permite afișarea caracterului %.

— Este posibilă utilizarea directă din programe C Aztec a funcțiilor BDOS de I/E, a funcțiilor BIOS de I/E, precum și a instrucțiunilor IN, OUT de acces la porturile de I/E.

– Zona de memorie alocată dinamic se află imediat după sfîrșitul programelor și datelor alocate static; ea se poate extinde pînă la stiva programelor C, care ocupă ultimii 1024 de octeți din zona TPA (dimensiunea stivei se poate modifica folosind funcția „rsvstk").

— Este posibilă redirectarea cererilor de I/E de la fișierele standard de intrare și de ieșire (asociate consolei) către oricare alte fișiere fără modificarea programului. De exemplu comanda

PROG <INP.DAT

are ca efect înlocuirea consolei cu fișierul INP.DAT în toate funcțiile de citire de la consolă (getchar, gets, scanf).

Comanda

PROG >OUT.DAT

are ca efect înlocuirea consolei cu fișierul OUT.DAT în toate funcțiile de scriere la consolă (putchar, puts, printf).

Este posibilă redirectarea simultană a intrării și ieșirii:

PROG <INP.DAT > OUT.DAT

— Depanarea programelor C se face în principal la nivelul limbajului C, iar in extremis la nivel de limbaj mașină.

Pentru depanarea codului mașină cu SID (ZSID) sînt prevăzute două funcții care permit apelarea programului depanator dintr-un program C: brkpnt ( ), tagpnt (tag).

Exemplu de folosire a funcției brkpnt:

```
SID TEST.COM TEST.SYM
#G          lansare program TEST
...
*nnnn apel brkpnt ( ) de la adresa nnnn
#X          comenzi SID de afișare registre și
#D,buf-     de afișare variabilă externă buf
#G          întoarcere din SID în programul TEST
```

8.2.3. Utilizarea de subrutine în limbaj mașină

Scrierea sau rescrierea unor părți de program în limbaj de asamblare poate fi necesară pentru reducerea lungimii programelor în cod mașină.

Cea mai comodă formă de combinare a celor două limbaje este scrierea unor funcții în limbaj de asamblare, asamblarea lor separată și legarea cu modulele obiect provenite din C în faza de linkeditare. În acest scop sînt necesare cîteva informații despre modul cum generează cod compilatorul C.

Compilatorul C Aztec adaugă la sfîrșitul fiecărui nume de funcție C un caracter '—' (subliniere și nu minus) atunci cînd trece numele respectiv în limbaj de asamblare.

Așadar un apel de funcție C de forma ex ( ) este tradus într-o instrucțiune CALL ex_.

Un apel de funcție C cu argumente generează o secvență de instrucțiuni care realizează punerea argumentelor în stiva adresată de registrul SP începînd cu ultimul argument și apoi apelează funcția. Deci la intrarea într-o funcție stiva conține următoarele (începînd cu adrese mici, deci de la vîrful stivei):

— adresa de revenire;
— valoarea primului argument;
— valoarea celui de al doilea argument;
— . . . .
— valoarea ultimului argument.

Scoaterea argumentelor din stivă se face tot de către funcția care a făcut apelul și deci care a pus argumentele în stivă; funcția apelată folosește argumentele primite în stivă, dar nu are voie să scoată din stivă argumentele. Mai precis, imediat înainte de revenirea dintr-o funcție (înainte de execuția instrucțiunii RET), registrul SP trebuie să aibă valoarea de la intrarea în funcție.

Registrele BC sînt utilizate de către compilatorul C Aztec pentru prima variabilă declarată de tip „register" (sau pentru primul argument

de funcţie declarat cu „register"); celelalte registre sînt folosite ca registre de lucru şi deci nu trebuie salvate de către fiecare funcţie care le foloseşte.

8.3. Funcţii standard pentru C Aztec

Pentru fiecare din funcţiile aflate în biblioteca LIBC.LIB sau SOFTLIBC.REL se dă antetul funcţiei şi declaraţia argumentelor precum şi o scurtă descriere a funcţiei.

Deşi o parte dintre funcţii sînt scrise în limbaj de asamblare, pentru ele modul de prezentare este acelaşi ca pentru funcţiile scrise în C.

8.3.1. Funcţii din bibliotecă LIBC (softlbc) scrise în C

char * *alloc* (size)
  unsigned size;

Alocare zonă continuă de memorie de lungime „size" octeţi; are ca rezultat adresa zonei alocate sau 0 (NULL) dacă nu există o zonă liberă de mărimea solicitată.

float *atof* (cp)
  char *cp;

Conversie numere reale din format extern (ASCII) în format intern; Argumentul „cp" este adresa şirului de caractere în format extern. Rezultatul este o valoare de tip "float"

atoi (cp)
  char *cp;

Conversie numere întregi din format extern (ASCII) în format intern; "cp" este adresa şirului de caractere în format extern. Rezultatul este o valoare întreagă

long *atol* (cp)
  char *cp;

Conversie întregi lungi din format extern (ASCII) în format intern "cp" este adresa şirului de cifre în format extern. Rezultatul este o valoare de tip „long".

close (fd)
  int fd;

Închidere fişier cu descriptorul "fd" (fişier deschis prin open sau creat); întoarce—1 în caz de eroare.

creat (name, mode)
  char *name; int mode;

Creare fişier cu numele extern "nume" în modul ."mode" şi deschidere fişier pentru scriere. Are ca rezultat un descriptor de fişier de tip "int"

sau —1 dacă nu este posibilă crearea fişierului (nu mai este loc în directorul discului)

exit (code)
unsigned code;

Terminare program şi închidere fişiere; dacă codul de terminare "code" este diferit de zero, atunci se şterge fişierul cu numele "$$$.SUB" (nu se mai execută alte comenzi din fişierul de comenzi în care se află programul terminat cu eroare).

fclose (fp)
FILE *fp;

Închidere fişier "fp" (deschis prin fopen)

fflush (fp)
FILE* fp;

Golire zonă tampon asociată fişierului „fp" prin scriere conţinut zonă tampon în fişier.

char * *fgets* (s, n, fp)
char *s; int n; FILE * fp;

Citirea unei linii de maximum "n" caractere din fişierul indicat de adresa "fp" într-un şir la adresa "s". Are ca rezultat adresa şirului creat în memorie (deci tot "s"), sau 0 la sfîrşit de fişier (la întîlnirea caracterului ^Z).

FILE * *fopen* (name, mode)
char *name, *mode;

Deschidere fişier cu numele extern "name" pentru modul de exploatare "mode". Are ca rezultat un pointer către structura de descriere a fişierului, numită FILE. Şirul "mode" poate avea valorile următoare: "r" (read) = numai citire, "w" (write) = numai scriere, "a" (append) = adăugare la sfîrşit de fişier, "r+" = citire sau scriere, "w+" = scriere şi citire, "a+" = adăugare şi citire. Dacă există un alt fişier cu numele „name", acesta este şters înainte de crearea noului fişier.

fprintf (fp, fmt, args)
FILE *fp; char *fmt; unsigned args;

Scriere cu format în fişierul „fp" (deschis prin fopen); „fmt" este şirul format, iar „args" este o listă de argumente separate prin virgule, ale căror valori se scriu în fişier.

fputs (s, fp)
char *s; FILE *fp;

Scriere şir de caractere terminat cu zero de la adresa „s" în fişierul „fp" (deschis prin fopen). În caz de eroare întoarce—1.

fread (buf, size, n, fp)
char *buf; unsigned size; int n; FILE *fp;

Citire la adresa „buf" a „n" unităţi de lungime „size" octeţi din fişierul

„fp" (deschis cu fopen); are ca rezultat numărul de unități citite sau 0 la sfîrșit de fișier.

fscanf (fp, fmt, args)
FILE *fp; char *fmt; int *args;

Citire cu format din fișierul „fp" (deschis cu fopen), conform șirului format „fmt" cu memorarea valorilor citite la adresele „args"; are ca rezultat numărul de valori citite sau —1 dacă s-a ajuns la sfîrșit de fișier.

fseek (fp, pos, mode)
FILE *fp; long pos; int mode;

Poziționare în fișierul „fp" (deschis prin fopen) la un număr de „pos" octeți față de începutul fișierului dacă „mod" = 0, față de poziția curentă, dacă „mod" = 1, sau față de sfîrșitul fișierului, dacă „mod" =2.

long *ftell* (fp)
FILE *fp;

Obținere poziție curentă în fișierul „fp" (deschis cu fopen) sub forma unui întreg lung care reprezintă numărul de octeți de la începutul fișierului pînă în poziția curentă.

fwrite (buf, size, n, fp)
char *buf; unsigned size; n; FILE *fp;

Scriere de la adresa „buf" a „n" unități de lungime „size" octeți în fișierul „fp" (deschis prin fopen); are ca rezultat numărul de valori scrise sau 0 în caz de eroare.

getbuf (fp)
FILE *fp;

Alocare zonă tampon pentru fișierul „fp" (deschis prin fopen); nu întoarce un rezultat direct. La funcția getbuf se poate face apel direct de programator sau este apelată implicit de getc și putc.

getc (fp)
FILE *fp;

Citire secvențială a caracterului următor din fișierul „fp" (deschis prin fopen); are ca rezultat octetul citit sau —1, dacă se citește după sfîrșitul fișierului (după ultima înregistrare a fișierului, nu se interpretează ^Z)

char **gets* (s)
char *s;

Citire șir de caractere din fișierul stdin la adresa „s"; în fișier șirul se termină cu CR, LF, iar în memorie șirul se termină cu zero; are ca rezultat adresa „s".

getw (fp)
FILE *fp;

Citire cuvînt (de 16 biţi) din fişierul ,,fp'' (deschis prin fopen); are ca rezultat cuvîntul citit (octetul inferior este primul caracter citit din fişier) sau —1 în caz de eroare.

long *lseek* (fd, pos, mode)
  int fd, mode; long pos;

Poziţionare în cadrul fişierului ,,fd'' (deschis prin open) la o distanţă de ,,pos'' octeţi faţă de începutul fişierului, dacă mod = 0, faţă de poziţia curentă, dacă mod = 1, sau faţă de sfîrşitul fişierului, dacă mod = 2; are ca rezultat (lung) poziţia ,,pos'' sau —1L în caz de eroare.

open (name, flag, mode)
  char *name; int flag, mode;

Deschidere fişier cu numele extern ,,name'' în modul indicat de ,,flag'' şi de ,,mode''; are ca rezultat un descriptor întreg de fişier ,,fd'' sau —1 în caz de eroare. Argumentul ,,mode'' poate fi: 0 = numai citire, 1 = numai scriere, 2 = citire şi scriere, 0x102, 0x202, 0x302 = şterge un fişier anterior cu acelaşi nume, 0×502 = se păstrează un fişier anterior cu acelaşi nume şi se întoarce —1.

posit (fd, pos)
  int fd; unsigned pos;

Poziţionare în cadrul fişierului ,,fd'' (deschis prin open) pe sectorul cu numărul ,,pos''; întoarce 0 la terminare cu succes şi —1 în caz de eroare.

printf (fmt, args)
  char *fmt; unsigned args;

Imprimare cu format a unei liste de valori la fişierul stdout; ,,fmt'' este adresa şirului format, ,,args'' este primul dintr-o listă de argumente separate prin virgule; este o funcţie fără rezultat direct

putc (c, fp)
  int c; FILE *fp;

Scriere secvenţială a caracterului ,,c'' în fişierul ,,fp'' (deschis prin fopen); are ca rezultat caracterul ,,c'' sau —1 în caz de eroare.

puts (s)
  char *s;

Scriere şir de caractere terminat cu zero de la adresa ,,s'' în fişierul stdout; are ca rezultat ultimul caracter scris sau —1 în caz de eroare.

read (fd, buf, len)
  int fd, len; char *buf;

Citire din fişierul ,,fd'' (deschis prin open) la adresa ,,buf'' a unui număr de ,,len'' octeţi; are ca rezultat numărul de octeţi citiţi sau —1 în caz de eroare.

rename (old new)
  char *old, *new;

Schimbare nume fişier din „old" în „new" (ambele nume sînt şiruri de caractere terminate cu zero); întoarce 0 la terminare cu succes şi —1 la terminare cu eroare.

scanf (fmt, args)
 char *fmt; int *args;

Citire cu format de la stdin: şirul format se află la adresa „fmt", „args" este primul dintr-o listă de variabile pointer care conţin adresele unde se depun valorile citite; are ca rezultat numărul de valori citite sau —1 la sfîrşit de fişier (^Z).

setbuf (fp, buf)
 FILE *fp; char *buf;

Stabilire adresă zonă tampon „buf" pentru fişierul „fp" (deschis prin fopen); se apelează înainte de prima citire sau scriere; nu se foloseşte împreună cu funcţia getbuf; nu întoarce un rezultat direct.

sprintf (s, fmt, args)
 char *s, *fmt; unsigned args;

Conversie de format în memorie: valorile argumentelor din lista „args" se convertesc în caractere conform şirului format „fmt" şi se depun începînd de la adresa „s".

sscanf (s, fmt, args)
 char *s, *fmt; int *args;

Conversie de format în memorie: se interpretează caracterele din şirul „s" conform şirului format „fmt", iar valorile rezultate din conversie se depun la adresele din argumentele listei „args".

ungetc (c, fp)
 char c; FILE *fp;

Anulare citire caracter cu funcţia „getc" prin înapoierea caracterului „c" în fişierul „fp" deschis prin „fopen"; prima apelare a funcţiei getc va furniza ca rezultat caracterul „c".

unlink (name)
 char *name;

Ştergere fişier cu numele extern „name"; întoarce 0 la terminare normală şi —1 în caz de eroare.

write (fd, buf, len)
 int fd, len; char *buf;

Scriere în fişierul „fd" (deschis prin creat sau open) de la adresa „buf" a unui număr de „len" octeţi; întoarce numărul de octeţi scrişi sau —1 în caz de eroare.

8.3.2. Funcţii din LIBC (SOFTLIBC) scrise în limbaj maşină

unsigned *bdos* (fct, arg)
unsigned fct, arg;

Apelare funcţie BDOS (CP/M) cu transmiterea codului funcţiei în „fct" (cod introdus apoi în registrul C) şi unui parametru în „arg" (introdus în registrele DE). Rezultatul funcţiei bdos este valoarea din registrele HL (şi A) şi depinde de codul funcţiei.

bdoshl (fct, arg)
unsigned fct, arg;

Apelare rutina BDOS, fără modificarea registrelor HL.

bios (cod, arg)
int cod, arg;

Apelare rutină BIOS (CP/M) cu transmiterea numărului rutinei în tabela de salturi BIOS (întreg cu valorile 0, 1, 2, . .) şi a unui parametru necesar executării rutinei BIOS. Rezultatul funcţiei rămîne în registrul A şi depinde de codul funcţiei.

bioshl (cod, arg)
unsigned cod, arg;

Apelare rutină BIOS fără modificarea registrelor HL.

boot ( )

Reiniţializare sistem CP/M. (echivalent cu bdos (0)).

brkpnt ( )

Apelare depanator (DDT, SID, ZSID) dintr-un program C (programul depanator se află deja încărcat în memorie); întoarcerea din depanator în program se face cu comanda „G".

clear (cp, len, val)
char *cp; unsigned len; int val;

Şterge zona de la adresa „cp" prin copierea caracterului „val" pe lungimea „len".

unsigned *CPM* (fct, arg)
unsigned fct, arg;

Identică cu funcţia bdos.

fcbinit (fname, fcb)
char *fname, fcb;

Iniţializare bloc de control fişier (fcb) CP/M: se transferă la adresa „fcb" numele discului, numele şi extensia fişierului din şirul „fname" (trecute în litere mari) şi completează cu zerouri ceilalţi octeţi.

char *in* (p)
char p;

Citire octet de la portul de intrare cu numărul „p"; are ca rezultat octetul citit.

char * *index* (s,c)
char *s,c;

Căutare caracter „c" în şirul „s"; are ca rezultat adresa primei apariţii a caracterului „c" în şirul „s".

isdigit (c)
char c;

Testare dacă caracterul „c" este o cifră zecimală; rezultatul este 1, dacă „c" este o cifră, şi 0, dacă „c" nu este cifră.

islower (c)
char c;

Testare dacă caracterul „c" este o literă mică; rezultatul este 1, dacă „c" este o literă mică între „a" şi „z" sau 0, dacă „c" nu este o literă mică.

isspace (c)
char c;

Testare dacă caracterul „c" este un spaţiu alb (blanc, tab, linie nouă); rezultatul este 1, dacă „c" este un spaţiu alb, şi 0, în caz contrar.

isupper (c)
char c;

Testare dacă caracterul „c" este o literă mare; rezultatul este 1, dacă „c" este o literă mare între „A", şi „Z" sau 0, în caz contrar.

out (p,c)
char p,c;

Scriere octet „c" la portul de ieşire cu numărul „p".

char * *rindex* (s,c)
char *s,c;

Căutare caracter „c" în şirul „s"; are ca rezultat adresa ultimei apariţii a caracterului „c" în şirul „s".

rsvstk (stk)
unsigned stk;

Modificare dimensiune stivă la valoare „stk"; dimensiunea implicită a stivei utilizate la execuţia programelor C este de 1024 octeţi.

char * *sbrk* (nb)
unsigned nb;

Cere „nb" octeţi din memoria alocată dinamic; întoarce ca rezultat adresa zonei alocate sau —1 dacă nu mai există suficientă memorie disponibilă.

char * *settop* (nb)
unsigned nb;

Cere „nb" octeţi din memoria alocată dinamic şi întoarce ca rezultat adresa zonei obţinute sau NULL (0) dacă nu mai există suficientă memorie disponibilă.

char * *strcat* (d,s)
  char *d, *s;

Adăugarea unei copii a şirului ,,s" la sfîrşitul şirului destinaţie ,,d"; are ca rezultat adresa şirului creat (tot ,,d").

strcmp (s,t)
  char *s, *t;

Comparare caracter cu caracter a şirurilor de la adresele ,,s" şi ,,t"; are ca rezultat un întreg <0, dacă primul şir este mai mic, = 0, dacă cele două şiruri sînt identice, şi >0, dacă primul şir este mai mare (comparaţie lexicografică).

char * *strcpy* (d,s)
  char *d, *s;

Copiere şir sursă ,,s" peste şirul destinaţie ,,d"; are ca rezultat adresa şirului ,,d".

strlen (s)
  char *s;

Calcul lungime şir de la adresa ,,s" (număr de caractere pînă la primul zero); are ca rezultat lungimea şirului ,,s".

char * *strncat* (d,s,n)
  char *s, *d; int n;

Adăugare la sfîrşitul şirului ,,d" a cel mult ,,n" caractere din şirul ,,s": primele ,,n" caractere din ,,s" sau toate caracterele din ,,s" pînă la primul octet zero; are ca rezultat adresa ,,d" a şirului creat.

strncmp (s, t, n)
  char *s, *t; int n;

Comparare şiruri de caractere pe lungimea ,,n": se compară primele ,,n" caractere (dacă nu se întîlneşte mai înainte un octet zero) din şirurile ,,s" şi ,,t"; are ca rezultat un întreg <0, dacă şirul ,,s" este mai mic decît şirul ,,t", = 0, dacă cele două şiruri sînt identice, şi > 0, dacă şirul ,,s" este mai mare decît şirul ,,t".

char * *strncpy* (d, s, n)
  char *d, *s; int n;

Copiere a primelor ,,n" caractere din şirul sursă ,,s" peste şirul destinaţie ,,d"; este posibil ca şirul rezultat să nu se mai termine cu zero; are ca rezultat adresa ,,d" a şirului nou creat.

unsigned *tagpnt* (tag)
  unsigned tag;

Apelare depanator (DDT, SID, ZSID) după încărcarea valorii ,,tag" în registrele HL; la revenirea în programul C (prin comanda ,,G") registrele HL îşi menţin conţinutul stabilit prin comenzi ale depanatorului.

tolower (c)
char c;

Conversie din litere mari în litere mici pentru caracterul „c"; rezultatul este litera mică corespunzătoare literei „c".

toupper (c)
char c;

Conversie din litere mici în litere mari pentru caracterul „c"; rezultatul este litera mare corespunzătoare literei „c".

8.3.3. Funcţii din biblioteca MATH LIB (soffmath. rel)

Pentru toate funcţiile care urmează, argumentele „x" şi „y" sînt de tip „double".

double *acos* (x)

Are ca rezultat arccos (x).

double *adx* (x, n)

Are ca rezultat pe x ridicat la puterea întreagă n.

double *amax1* (x, y)

Are ca rezultat valoarea cea mai mare dintre x şi y.

double *asin* (x)

Are ca rezultat arcsin (x).

double *atan* (x)

Are ca rezultat arctg (x).

double *atan2* (x, y)

Are ca rezultat arctg (x/y).

double *cos* (x)

Are ca rezultat cos (x).

double *cosh* (x)

Are ca rezultat cosinus hiperbolic de x.

double *cotan* (x)

Are ca rezultat cotangentă de x.

double *exp* (x)

Are ca rezultat numărul „e" la puterea x (exponenţială).

double *fabs* (x)

Are ca rezultat valoarea absolută a lui x.

double *log* (x)

Are ca rezultat ln (x), deci logaritm natural de x.

double *log10* (x)

Are ca rezultat log (x), deci logaritm în baza 10 de x.

double *pow* (x, y)

Are ca rezultat pe „x" ridicat la puterea „y".

double *ran* ( )

Are ca rezultat un număr pseudoaleator subunitar.

double *randl* (x)

Are ca rezultat un număr pseudoaleator dintr-o distribuţie exponenţială.

double *sign* (x, y)

Are ca rezultat valoarea „x" cu semnul lui „y".

double *sin* (x)

Are ca rezultat sin (x).

double *sinh* (x)

Are ca rezultat sinusul hiperbolic de x.

double *sqrt* (x)

Are ca rezultat rădăcina pătrată din valoarea x.

double *tan* (x)

Are ca rezultat tg (x).

double *tanh* (x)

Are ca rezultat tangenta hiperbolică de x.

8.4. Exemple de programe C

1) Valoarea absolută a unui număr întreg

```
abs2 (n)
register int n;
 { if (n < 0)
     return -n ;
   return n ;
 }
```

```
abs (n)
int n;
{ register int nn;
  if ( (nn=n) < 0)
   return -nn ;
  else
   return nn ;
```

2) Cîtul împărţirii a doi întregi prin scăderi succesive

```
idiv (a,b)
int a,b ;
{ register int c ;
 c = 0 ;
 while (a >= b)
   { a =a-b ; c =c+1 ;}
 return c ;
}
```

```
idiv2 (a,b)
int a ,b;
{ register int c;
 for (c=0; a >=b; c++)
     a -= b ;
 return c;
}
```

3) Căutarea secvenţială a unei valori date „x" într-un tablou „a" de „n" numere întregi. Rezultatul acestei funcţii este fie poziţia valorii căutate în lista dată, fie —1 dacă 'x' nu se află în lista „a"

```
cauta (x,a,n)
int x,n,a[];
{ int i;
 for (i=0; i<n; i++)
  if (x==a[i])
   break;
 if (i == n)
  return -1;
 else
  return i;
}
```

```
cauta2 (x,a,n)
int x,n,a[];
{ int i;
i=0;
 while (i<n &&  x != a[i])
   ++i;
   if (x==a[i])
  return i;
else
  return -1;
```

Următoarele variante folosesc faptul că instrucţiunea de ieşire din funcţie „return" produce şi ieşirea din ciclu:

```
cauta3 (x,a,n)
int x,n,a[];
{ int m;
for (m=n-1; m>=0; m--)
  if (x==a[m])
   return m;
return -1;
```

```
cauta4 (x,a,n)
int x,n,a[];
{while (n)
   if (x==a[--n])
      return n;
return -1;
}
```

4) Extragerea rădăcinii pătrate dintr-un număr real x

```
#define EPS 1.e-5
double sqrt (x)
double x;
{ double r1, r2;
if (x<0) return -1;
if (x==0) return 0;
r2=x/2.0;
 do
  { r1=r2; r2=(r1+x/r1)/2.0;}
while (fabs(r2-r1) > EPS);
return r2;
}
```

5) Ordonarea unui tablou t de numere întregi prn metoda inversării elementelor vecine

```
nsort (t,n)
int t[],n;
{ int cinv,i,tt;
do {
  cinv=0; /* contor de inversari */
  for(i=0;i<n-1;i++)
   if(t[i+1]<t[i])
    {cinv++; tt=t[i]; t[i]=t[i+1]; t[i+1]=tt;}
} while (cinv);
}


main ()   /* program de test a functiei nsort */
{ int list[50],m,i;
m=0;
while ( scanf("%4d",&list[m]) != -1)
  ++m;
--m; nsort (list,m);
for(i=0;i<m;i++)
printf("%6d",list[i]);
```

6) Căutarea unui caracter c într-un șir s; întoarce ca rezultat adresa primei apariții a lui c în șirul s sau zero, dacă c nu se află în șirul s.

```
char *index(c,s)
char c,*s;
{ register int found; /* indicator de iesire din ciclu */
found=0;
while (!found)
 if(c == *s) found=1;
 else ++s;
return found? s : 0;
}
```

Varianta 2:

```
char index2 (c,s)
char c, *s;
{
while (*s)
 if(c== *s++)
  break;
return *s.? --s : 0 ;
}
```

Varianta 3.

```
}
char *index3 (c,s)
char *s,c;
{ register char *rs;
for(rs=s;*rs;++rs)
 if(*rs==c)
    return rs;
return 0;
}
```

7) Copierea unui şir de caractere terminat cu zero

```
..cpy (d,s)
char d[], s[],
{ int i,
for (i=0; s[i]; i++)
   d[i]=s[i];
}
```

Alte variante de copiere:

```
strcpy2 (d,s)
char *d. *s;
{ while (*s !='\0')
    *d++=*s++;
}
```

```
strcpy3 (d,s)
char *d. *s;
{ while (*d++=*s++)

}
```

8) Conversie număr întreg din format extern în format intern

```
atoi (ascii)
char ascii[];
{ int i, j, val; char semn;
for (i=0;iswhite(ascii[i]); i++)
   ;   /* ignora spatii albe */
if ((semn=ascii[i])=='+' !! semn=='-')
   i++;
val=0;
for (j=i; isdigit(ascii[j]); ++j)
   val=10*val +ascii[j] -'0';
if (semn=='-')
   return (-val);
return val;
}
```

Altă variantă:

```
atoi (adr)
char *adr;        /* adresa  sirului de caractere */
{ register int val;
char c, semn;
while (iswhite(*adr))
 ++ adr ;         /* ignora spatii alb */
switch (*adr) {
  case '-': semn=1; ++ adr; break;
  case '+': ++adr;
  default:  semn=0;
}
for (val=0; isdigit(c=*adr++); )
  val = val*10 +c-'0';
return semn? -val : val ;
}
```

9) Concatenarea a două şiruri într-un singur şir

```
strcat (d,s)
char d[], s[];
{ int i, j;
while (d[i] != '\0')
  i++;  /* cauta sfirsitul sirului d */
while (( d[i++]=s[j++]) != '\0')
  ;    /* copiaza */
return &d ; /* eventual */
}
```

Varianta cu pointeri:

```
strcat (d,s)
char *d, *s;
{ char *dd;
for (dd=d; *dd; ++dd)
  ;   /* pozitionare pe sfirsit de sir */
while (*dd++=*s++)
  ;   /* copiaza  sirul s dupa sirul d */
return d;
}
```

10) Căutarea unui şir t într-un alt şir s; rezultatul este adresa şirului t în şirul s sau —1 dacă t nu este un subşir din s

```
sindex (s,t)
char s[], t[];
{ int i, j, k;
for (i=0; s[i] != '\0'; i++)
 { for (j=i,k=0; t[k] !='\0' && s[j]==t[k];j++,k++)
      ;
   if (t[k]=='\0') return i;
 }
return -1;
}
```

Funcţie pentru determinarea poziţiei unui şir s1 într-un alt şir s2 folosind funcţii standard pe şiruri de caractere

```
char * pos(s1,s2)
char *s1, *s2;
{
char *s, *p;
char *index();
s=s2;
while (p=index(s,*s1))
  if(strncmp(s1,p,strlen(s1))==0)
   return p;
  else
   s=p+1;
return 0;
}
```

11) Imprimarea argumentelor din linia de comandă

```
main (argc,argv)
int argc;  /* numar de argumente */
char *argv[];  /* tablou cu adresele argumentelor */
{ int i;
for (i=1;i<argc;i++)
  { puts (argv[i]); putchar ('\n');}
}
```

Varianta următoare foloseşte indirectarea pentru parcurgerea tabloului de pointeri argv (argv este aici un pointer către adresa şirului argument următor şi nu un nume de tablou).

```
main (argc,argv)
int argc; char **argv;
{ while (--argc)
  { puts(*++argv); putchar('\n');}
}
```

12) Citirea unui şir de caractere în memorie; rezultatul este lungimea şirului citit (0 pentru un şir nul)

```
getstr (adr,lmax)
char *adr;      /* adresa unde se citeste sirul */
int   lmax;        /* lungimea maxima a sirului citit */
{ int c,i;
i=0;
while (--lmax > 0 && (c=getchar () != EOF)
  { if (c == '\n') break;
    *adr++ = c; i++ ;
  }
*adr=0;   /* terminator de sir */
return i;
}
```

13) Conversia unui număr întreg n din forma internă (binară) în caractere ASCII interpretat în baza „base". Caracterele se pun începînd de la adresa „asc".

```
static char digits[]="0123456789abcdef";
convert(n,base,asc)
register unsigned n;
int base;
char *asc;
{ do
    {*asc++ = digits [(int)(n % base)];}
  while ( n /= base) != 0);
}
```

14) Ordonarea unei liste de caractere prin crearea și parcurgerea infixată a unui arbore binar de sortare

```
struct nod
{ char val;  /* valoare nod */
  struct nod *left; /*  succ stinga */
  struct nod *right; /*  succ dreapta*/
};
char *alloc();

infix(p)  /* traversare infixata arbore binar */
struct nod *p;
{ if (p != 0)
  { infix (p->left);
    putchar(p->val);
    infix (p->right);
  }
}
adaug (rad,c)  /* adaugare nod la un arbore binar */
struct nod *rad;
char c;
{ struct nod *p, *q, *n;
 p=rad;  /* se pleaca dela radacina */
 while (p != NULL)
  {q=p;
   if (c == p->val)
     return -1;    /* nodul exista deja in arbore */
   if (c < p->val)
    p = p->left;  /* mergi la stinga */
   else
    p = p->right; /* mergi la dreapta */
  }
 if ((n=alloc(sizeof(*p) )) == NULL)
    return -1;  /* nu mai este loc in memorie */
 /* legare nod de la adresa n la arbore */
 n->val = c; n->left = n->right = NULL;
 if (c < q->val)
  q->left = n ;
 else
  q->right = n;
 return 0;
}
```

```
main ()
{ int c ;
 struct nod *p;
 /* radacina arbore */
 scanf("%s",&c); p=alloc(sizeof (*p));
 p->val=c; p->left = p->right =NULL;
 /* citire si adaugare noduri */
 while (scanf("%s",&c) != -1)
   if(adaug (p,c) == -1)
     break;
 infix (p);
}
```

15) Program pentru compararea octet cu octet a două fișiere; numele celor două fișiere se dau în linia de comandă astfel: cmp fișier1 fișier2

```
main (argc,argv)
int argc; char *argv[];
{ FILE *f1, *f2;
int c1, c2; long n;
f1=fopen (argv[1],"r");
f2=fopen (argv[2],"r");
if (f1==NULL || f2==NULL)
 {puts ("Eroare la deschidere fisiere\n");
  return;}
n=1;
while ( (c1=getc(f1)) == (c2=getc(f2)) )
 {if ( (c1 == EOF) || (c2 == EOF) ) break;
  ++n;
 }
if(c1==EOF && c2==EOF)
 puts ("Fisiere identice\n");
else
 printf ("Fisierele difera la octetul nr. %lu",n);
fclose(f1); fclose(f2);
}
```

16) Căutarea și imprimarea liniilor dintr-un fișier text care încep cu un șir dat.

```
#define MAXL 128
main ()
{ FILE *f;
 char linie [130], numef[15], text[30];
 puts ("\n Nume fisier:"); gets (numef);
 puts ("\n Sirul cautat:"); gets (text);
 if ( (f=fopen(numef,"r")) == NULL)
  { puts ("\n ? Fisier inexistent");
    return; }
 puts ("\Linii care incep cu sirul dat:\n");
 while ( fgets(linie,MAXL,f) == &linie )
   if (strncmp (text,linie,strlen(text))==0 )
    puts (linie);
```

17) Funcții de creare și de listare a unui fișier de numere reale redrezentînd valorile unei funcții.

```
/* func''* de citire si listare fisier */
fr ()
{ float x[2];
  if ((f=fopen(numef,"r"))==NULL)
   return  ;
 while (fre d(&x 4 ,2,f))
  printf ("x=%  2*x=%f\n",x[0],x[1]);
 fclose (f);return 1;
}

char numef [15]; FILE *f;
/* functie de scriere date in fisier */
fw ()
{ float x,y;
 if ((f=fopen(numef,"w"))==NULL)
  return -1;
 for (x=0.0;x<10.;x+=0.1)
  {y=2.*x;
   fwrite(&x,sizeof x,1,f);
   fwrite(&y,sizeof x,1,f);
  }
fclose (f); return 1;
}
```

18) Copierea unui fișier bloc cu bloc, folosind un fișier temporar de lucru

```
main(argc,argv)
char *argv[];
{
char buf[BUFSIZ];
int n,id,od;
if ((id=open(argv[1],0)) == -1)
 { puts ("Nu exista fisierul sursa\n"); return; }
if ((od=creat("COPY.$$$")) == -1)
 { puts ("Nu se poate crea fisierul destinatie\n");
  return;}
while ((n=read(id,buf,BUFSIZ)) > 0)
     write(od,buf,n);
close(id); close(od);
  unlink(argv[2]);
  rename ("COPY.$$$",argv[2]);
}
```

19) Citire și scriere în acces direct pe baza unui număr de înregistrare în fișier (înregistrări de 128 octeți)

```
main ()
 {char opt[2] ; int df, i ; char bloc[128];
 unsigned ns;
 long lseek();
```

```
 if ((df=creat("SEEK.$$$"))==-1)
  puts ("! directory full");
/* initializare fisier ;scriere secventiala */
 clear (bloc,128,'?');
 for (i=10;i>0;i--)
  write (df,bloc,128);
/* scriere si citire in acces direct */
 df=open("SEEK.$$$",2);
 do {
  clrscr(); clear (bloc,128,' ');
  puts("Rs = citire sector s\n");
  puts("Ws = scriere sector s\n");
  puts("E = terminare program");
  scanf ("%1s",&opt);
  switch (*opt) {
     case 'R': scanf("%4d",&ns);
               lseek(df,(long) 128*ns,0);
               if(read(df,bloc,128)==-1)
                {printf("\neroare la citire"); break;}
               printf("%.80s\n",bloc);
               wait ();break;
     case 'W': scanf("%4d",&ns);
               lseek(df,(long) 128*ns,0);
               printf("\ndate :");
               scanf("%128s",bloc);
               if(write(df,bloc,128)==-1)
                printf("\neroare la scriere");
               break;
      default : close(df); return;
    }
 }while (1);
}
     /* sterge ecran */
clrscr ()
{printf("\033I");}
     /* asteapta tastarea unui caracter */
wait ()
{char dummy[2];
scanf ("%1s",dummy);
}
```

20) Program de adăugare articole la sfirşitul unui fişier cu articole de lungime fixă, diferită de 128 octeţi. Se caută prima poziţie liberă din fişier (după ultimul articol scris) ştiind că ultima înregistrare este completată la închidere cu caracterul CTRL/Z (1AH)

```
#define LART 20      /* lungime articol */
main (argc,argv)
char *argv[]; int argc;
{
char art[128];  /* buffer ptr un articol de date */
char fdat [15];
```

```
strcat(strcpy(fdat,argv[1]),".DAT");
if(fopen(fdat,"r") ==0)
  fclose(fopen(fdat,"w"));
clr (art,128);            /* sterge zona buffer */
while (scanf("%20s",art) != -1)
 { adaug(fdat,art,LART);
   clr (art,128);
 }
}

 /* functie de adaugare articol */
adaug (fdat,bufart,lart)
char  bufart[], *fdat;
int lart;
{
char buf[128];  /* buffer ptr un articol de date */
FILE *fd;
int i;
long poz;
 fd=fopen(fdat,"a+");
 if(poz=ftell(fd))
  {poz -=128;
   fseek(fd,poz,0);
   fread(buf,128,1,fd);              /* citeste ultimul sector */
   for(i=127;buf[i]==0x1a;--i)
      ;
   poz=poz+(long)i;                  /* prima pozitie libera */
   fseek(fd,poz,0);
 }
 fwrite(bufart,lart,1,fd); /* scrie in fisier date */
 fclose(fd);
return;
}
clr (cp,size)
char *cp;int size;
{while(size--)
 *cp++=0;
      }
```

21) Exemplu de utilizare subrutină în limbaj mașină

```
main ()
{
char buf[51]; int c;
buf[50]='\0';
while ((c=bdos(1)) !=-1)
 { fill (&buf,50,c);
   puts (buf);
 }
}

;       Fisierul cu subrutina FILL_
          PUBLIC  FILL_
FILL_:    PUSH    B
          LXI     H,8
```

```
        DAD     SP          ;HL=adresa arg3
        MOV     A,M         ;A=arg.3 (valoare octet)
        DCX     H
        MOV     B,M         ;BC=arg.2 (lungime)
        DCX     H
        MOV     C,M
        DCX     H
        MOV     D,M         ;DE=arg.1 (adresa)
        DCX     H
        MOV     E,M
        XCHG                ;HL=adresa
        MOV     D,A         ;D=valoare octet
FILLP:
        MOV     A,B         ;este BC zero ?
        ORA     C
        JZ      FILEX
        MOV     M,D         ;pune octet in memorie
        INX     H           ;adresa urmatoare
        DCX     B           ;numar de octeti
        JMP     FILLP
FILEX:
        POP     B
        RET
        END
```

Iată şi o altă soluţie de preluare a argumentelor de către funcţia „fill":

```
;UMPLERE ZONA DE MEMORIE CU UN CARACTER
     PUBLIC    FILL_
FILL_:
       PUSH B      ;salvare BC
       LXI  H,4
       DAD  SP     ;HL=adResa arg.1
       MOV  E,M    ;DE=ARG.1
       INX  H
       MOV  D,M
       INX  H
       MOV  C,M    ;BC=arg.2
       INX  H
       MOV  B,M
       INX  H
       MOV  L,M    ;L=arg.3
       XCHG        ;E=valoare octet, HL=adresa
FILLP: MOV  A,B
       ORA  C
       JZ   FILEX
       MOV  M,E    ;pune octet in memorie
       INX  H      ;adresa urmatoare
       DCX  B      ;contor de octeti
       JMP  FILLP      ;repeta
FILEX: POP  B
       RET
```

22) Program segmentat cu o rădăcină și două segmente, care se încarcă unul pe altul în continuare (la adrese succesive)

```
      /*   Fisierul ROOT.C  */
int glbl;
main ()
{
int i; char c='A';;
settop (0x3000);
for (i=0;i<4;i++)
 { glbl=i;c++;
   printf ("\nmain %d",glbl);
   ovloader ("OVL1",c );
   ovloader ("OVL2",c );
 }
}

     /*    Fisierul OVL1.C    */
extern int glbl;
ovmain (ch)
char ch;
{
 printf ("\n  ovl1  %d  %c",glbl,ch);
}

     /*   Fisierul OVL2.C   */
extern int glbl;
ovmain (c)
char c;
{
 printf ("\n  ovl2  %d %c",glbl,c);
}
```

Observații:

— fiecare dintre cele trei segmente va scrie numele său și valorile variabilei externe „glbl" și parametrului „c":

— adresele și lungimele celor trei segmente afișate de LN sînt:

ROOT 0x0100 0x28C4
OVL1 0x29C4 0x7E
OVL2 0x2A42 0x68

— adresa cea mai mare (după OVL2) este 2AAA, dar în settop s-a utilizat o adresă ceva mai mare (2B00).

— comenzile necesare linkeditării modulelor obiect pentru cele trei segmente sînt:

LN —R ROOT.O OVLOADER.O LIBC.LIB
LN —R OVL1.O OVBGN.O ROOT.RSM LIBC.LIB
LN OVL2.O OVBGN.O OVL1.RSM LIBC.LIB

.23) Program segmentat cu o rădăcină și două segmente paralele suprapuse (încărcate alternativ după rădăcină la aceeași adresă)

```
    /*   Fisierul ROOT.C   */
int glbl;
main ()
{
int i; char c='A';;
settop (0x3000);
for (i=0;i<4;i++)
 { glbl=i;c++;
   printf ("\nmain %d",glbl);
   ovloader ("OVL1",c );
   ovloader ("OVL2",c );
 }
}

    /*   Fisierul OVL1.C   */
extern int glbl;
ovmain (ch)
char ch;
{
 printf ("\n  ovl1  %d  %c",glbl,ch);
}

    /*   Fisierul OVL2.C   */
extern int glbl;
ovmain (c)
char c;
{
 printf ("\n  ovl2  %d %c",glbl,c);
}
```

9. PROGRAMARE ÎN LIMBAJUL PASCAL SUB CP/M

Limbajul PASCAL şi-a cîştigat un loc sigur printre limbajele moderne de programare, fiind disponibil pe majoritatea echipamentelor de calcul, de la calculatoare personale la sisteme cu putere mare de calcul.

Pascal MT+ este un sistem complet de programe care permite folosirea unei variante extinse a limbajului Pascal pe orice microcalculator care funcţionează sub sistemul CP/M (spre deosebire de Turbo-Pascal, utilizabil numai pe maşini cu Z80).

Avantajul principal al acestui mediu de programare Pascal îl constituie tocmai îmbinarea dintre verificările permise de limbajul Pascal asupra programelor sursă şi facilităţile numeroase adăugate limbajului, care fac din Pascal MT+ un instrument practic de lucru, foarte apropiat ca posibilităţi de limbajul C, şi nu numai un limbaj didactic pentru programare disciplinată, structurată.

Limbajul Pascal-80 extinde mult standardul ISO Pascal atît prin adăugări la sintaxă, cît şi printr-un număr mare de funcţii predefinite.

Pascal MT+ este un exemplu tipic de sistem Pascal modern, atît din punct de vedere al limbajului, cît şi al posibilităţilor de lucru oferite: multe opţiuni de compilare, compilări separate, multe proceduri şi funcţii predefinite, cuplare cu subrutine în limbaj de asamblare şi inserare directă de cod maşină, vizualizare cod obiect generat, depanare simbolică, facilităţi pentru segmentarea programelor mari.

Dintre opţiunile de compilare menţionăm cîteva mai importante: recursivitate controlabilă pe porţiuni, generare de cod optimizat pentru Z80, reducerea dimensiunilor codului generat, verificări opţionale la execuţie, includere de fişiere la compilare ş.a.

Limbajul Pascal-80 este unul dintre cele mai puternice limbaje existente pe microcalculatoare, atît prin posibilităţile de prelucrare internă (operaţii pe şiruri de caractere, operaţii la nivel de bit, numere reale, adrese), cît şi prin posibilităţile de lucru cu fişiere disc, avînd faţă de limbajul C avantajul că este mult mai protejat la erori.

Utilizarea practică a limbajului Pascal-80 pune în evidență și unele inconveniente. Astfel, limitările impuse de mașină fac ca timpul de compilare plus linkeditare să fie destul de mare. Din același motiv, semnalarea și recuperarea din erori sintactice lasă de dorit în Pascal MT+ și practic se detectează numai prima eroare dintr-un program sursă. De asemenea, codul obiect generat nu este foarte eficient.

Pentru învățarea limbajului Pascal sau pentru dezvoltarea rapidă de programe relativ scurte se recomandă programul Turbo-Pascal, care se distinge printr-un timp de răspuns foarte bun.

Limbajul Turbo-Pascal este tot o extindere a limbajului Pascal standard, cu facilități de multe ori similare celor din Pascal MT+, dar cu altă formă sintactică.

Acest capitol prezintă numai particularitățile de programare și de utilizare a limbajului Pascal MT+, considerind că se cunoaște limbajul standard Pascal din alte lucrări.

9.1. Utilizarea sistemului de programe Pascal MT+

9.1.1. Utilizarea compilatorului Pascal MT+

Fișierele necesare pentru utilizarea limbajului PASCAL/MT+ sînt următoarele:

PASCAL.COM (sau MTPLUS.COM) prima parte a compilatorului
MTPLUS.000, MTPLUS.001, MTPLUS.002, MTPLUS.003, MTPLUS.004, MTPLUS.005 } segmente ale compilatorului
LINKMT.COM editor de legături
MTERRS.TXT lista mesajelor de eroare la compilare
PASLIB.ERL biblioteca relocabilă Pascal/MT+
DEBUGGER.ERL depanator simbolic
Fișierele următoare sînt utile, dar nu sînt absolut necesare:
FPREALS.ERL, TRANCEND.ERL, RANDOMIO.ERL,FULLHEAP.ERL } biblioteci (relocabile)
LIBMT.COM bibliotecar pentru format relocabil ERL
DIS8080.COM dezasamblor pentru codul generat de compilator

În forma sa cea mai simplă, operarea sistemului PASCAL/MT+ necesită următoarele etape:

— Compilarea programului din fișierul sursă TEST.PAS:

```
PASCAL TEST
```

Rezultatul compilării este un fișier relocabil TEST.ERL.

— Linkeditarea programului din fișierul relocabil cu biblioteca PASLIB:

LINKMT TEST,PASLIB/S

— Executarea programului din fişierul TEST.COM:

TEST

Observaţii

— am presupus că toate fişierele se găsesc pe un acelaşi disc, aflat pe unitatea implicită. Este posibil să se utilizeze un disc pentru compilări şi alt disc pentru linkeditare. Numele fişierelor pot fi precedate de numele discului suport;

— compilatorul consideră pentru fişierele sursă tipul implicit .PAS sau .SRC, dar se poate folosi orice tip;

— linkeditorul consideră pentru fişierele de intrare tipul implicit .ERL („Extended Relocatable") şi nu admite un alt tip;

— în comanda LINKMT pot apărea mai multe fişiere de intrare şi (sau) mai multe biblioteci, dar biblioteca PASLIB trebuie să fie ultima în linia de comandă;

— numele fişierului de tip .COM este acelaşi cu numele primului fişier de tip .ERL (programul principal), dar poate fi specificat explicit ca în comanda următoare:

LINKMT TEST = TEST,PASLIB/S

— opţiunea /S utilizată cu numele unei biblioteci semnifică o extragere selectivă din biblioteca respectivă a modulelor necesare şi nu includerea întregii biblioteci în programul creat.

Celelalte *biblioteci* de module relocabile din sistemul Pascal/MT+ au următoarea utilizare:

FPREALS.ERL bibliotecă aritmetică pentru numere reale în binar virgulă mobilă, inclusiv intrări-ieşiri pentru reali;

TRANCEND.ERL bibliotecă de funcţii transcendentale: funcţii trigonometrice, exponenţială etc.; foloseşte rutine din biblioteca FPREALS şi de aceea trebuie să o preceadă în linia de comandă pentru linkeditare (LINKMT parcurge o singură dată fişierele de intrare, în ordinea întîlnirii lor în comandă);

RANDOMIO.ERL bibliotecă de proceduri pentru acces direct la fişiere disc (acces „aleatoriu");

FULLHEAP.ERL bibliotecă cu procedurile standard NEW, DISPOSE de gestiune dinamică a memoriei. Biblioteca PASLIB.ERL conţine alte două proceduri NEW şi DISPOSE pentru o variantă mai simplă de alocare a memoriei dinamice, fără reutilizarea ei.

ROVLMGR.ERL o altă variantă a modului de gestiune a segmentării („Overlay Manager") decît cea din PASLIB.

Opţiuni de compilare în linia de comandă

Compilatorul Pascal/MT+ acceptă în linia de comandă, după numele fişierului de intrare un şir de parametri de compilare care începe cu caracterul „$" şi continuă pînă la sfîrşitul liniei. Se pot utiliza sau nu spaţii (blancuri) între parametrii succesivi.

Un parametru constă dintr-o literă sau o literă urmată de un alt caracter ce codifică un dispozitiv CP/M. Codificarea folosită este următoarea:

X = consola sistem (CON:)
P = imprimanta (LST:)
@ = unitatea de discuri implicită
A,B, . . = unitatea de discuri 0,1, . .

Lista alfabetică a opțiunilor de compilare:

A apelare automată LINKMT după compilare (este necesar un fișier de comenzi .CMD cu numele fișierului sursă). Implicit se revine în sistem după compilare.

B numerele reale se reprezintă în BCD (binar-zecimal). Implicit realii se reprezintă în binar virgulă mobilă.

C continuă compilarea în caz de eroare sintactică. Implicit compilatorul se oprește la fiecare eroare și întreabă dacă se continuă sau nu compilarea.

D generează informații pentru depanator în codul obiect și scrie un fișier de tip .PSY pe același disc cu fișierul .ERL. Implicit opțiunea este inactivă (nu se generează fișier PSY)

Ed fișierul MTERRS.TXT se află pe discul „d" (d = @,A,B,..). Implicit fișierul este căutat pe discul curent.

Od fișierele segmente MTPLUS.00n se află pe discul „d". Implicit toate segmentele pe discul curent.

Pd scrie fișierul de tip .PRN pe dispozitivul „d" (d = X,P,@,A,B). Implicit nu se generează fișier .PRN (lista de compilare).

Q nu se afișează mesaje la consolă pe parcursul compilării. Implicit compilatorul trasează desfășurarea compilării.

Rd scrie fișierul de tip .ERL pe discul „d". Implicit fișierul .ERL se scrie pe discul cu fișierul sursă.

Td scrie fișierul de manevră PASTEMP.TOK pe discul „d". Implicit fișierul de lucru pe discul curent.

V afișare nume de proceduri și funcții pe măsura compilării. Implicit nu se afișează numele procedurilor în faza 1.

X include în fișierul .ERL informații pentru dezasamblor. Implicit nu se generează aceste date în formatul .ERL

Z generează cod optimizat pentru procesorul Z80. Implicit generează numai instrucțiuni 8080.

@ interpretează caracterul „@" ca sinonim pentru „↑". Implicit caracterul „@" este tratat ca literă.

Exemple de utilizare opțiuni de compilare:

```
PASCAL TEST $QCZ
PASCAL TEST $RB PX D
```

Compilatorul Pascal/MT+ afișează în fiecare fază următoarele:
În faza 0 (de analiză sintactică):

— cîte un caracter „+" la fiecare 16 linii sursă și numărul total de linii sursă;

În faza 1 (construirea tabelului de simboluri):
— memoria disponibilă la începutul acestei faze;
— mărimea tabelului de simboluri utilizator;
— cîte un caracter „#" la fiecare funcție sau procedură;
— memoria disponibilă la sfîrșitul fazei 1;

În faza 2 (generarea de cod obiect):
— numele fiecărei proceduri și lungimea codului generat;
— coduri și (dacă există fișier MTERRS.TXT) mesaje de eroare;
— număr de linii sursă compilate;
— număr de erori detectate;
— număr de octeți de cod (în zecimal);
— număr de octeți de date (în zecimal).

Opțiuni de compilare incluse în programe Pascal/MT+.

Aceste opțiuni se introduc sub formă de comentarii speciale, care încep cu caracterul „$":

{$p} sau (* $p *)

unde „p" este numele parametrului (o literă), urmat eventual de „+" (opțiune activă) sau „—" (opțiune inactivă) sau de o valoare specifică fiecărei opțiuni de compilare.

Într-un comentariu se poate introduce o singură opțiune de compilare.

Lista alfabetică a opțiunilor de compilare-comentariu.

Cn generare de instrucțiuni RST n pentru apelare subrutine de lucru cu numere reale. Implicit se generează instrucțiuni CALL.

E+/E— generează sau nu atributul de „punct de intrare" pentru toate variabilele globale și numele de proceduri/funcții. Implicit E+, deci toate sînt simboluri externe.

Inume include la compilare fișierul sursă cu numele indicat, în locul unde se află opțiunea.

Kn elimină o serie de nume de proceduri standard din tabelul de simboluri predefinite, în funcție de valoarea lui n (0 — 15). Implicit tabelul de simboluri conține toate numele de funcții și de proceduri predefinite în Pascal MT+.

L+/L— activează (dezactivează) listarea programului sursă în faza 1 (se poate utiliza repetat pentru listarea selectivă). Implicit L+, deci se listează tot programul sursă.

P inserează caractere FF (Form Feed = 0CH) în fișierul .PRN.

R+/R— generează sau nu cod suplimentar pentru verificarea la execuție a depășirii indicilor de tablou și a subdomeniului pentru variabile de tip m .. n. Implicit R—, deci nu se fac aceste verificări la execuție.

Qn generare de instrucțiuni RST n pentru încărcări și memorări în proceduri recursive. Implicit se generează instrucțiuni CALL.

S+/S— alocare de memorie în stivă (S+) sau alocare statică (S—) pentru parametrii şi pentru variabilele locale din proceduri. Procedurile recursive trebuie compilate cu S+. Implicită este opţiunea S—, deci proceduri nerecursive.

T+/T— verifică sau nu corespondenţa strictă a tipurilor de date la execuţie, deci respectarea standardului. Implicită este opţiunea T—, deci varianta mai tolerantă.

X+/X— verifică sau nu la execuţie excepţiile aritmetice: împărţire prin zero la întregi şi la reali, depăşire superioară sau inferioară la operaţii cu reali, depăşire lungime maximă la şirurile de caractere (tipul STRING). Implicită este opţiunea T+, deci se fac verificările.

W+/W— verificare portabilitate (respectare standard) cu mesaje de avertisment (Warning), în caz de nerespectare. Se foloseşte împreună cu opţiunea T+. Implicită este opţiunea W—, deci nici o verificare.

Z$nnnn Iniţializare registru de stivă SP cu adresa hexa „nnnn".

Implicit registrul SP este încărcat cu adresa de sfîrşit a zonei TPA, luată din locaţiile 6 şi 7.

Observaţii:

— opţiunile E,K şi S trebuie folosite înaintea cuvintelor cheie PROGRAM sau MODULE (chiar la începutul textului PASCAL);

— opţiunea Z trebuie pusă în programul principal, înainte de BEGIN;

— opţiunile R,X,T,W pot fi utilizate de mai multe ori într-un acelaşi program cu valori „+" şi „—" pentru activarea şi dezactivarea selectivă pe porţiuni a verificărilor respective;

— interpretarea valorilor lui „n" la opţiunea Kn este:

0 ROUND, TRUNC, EXP, LN, ARCTAN, SQRT, COS, SIN
1 COPY, INSERT, POS, DELETE, LENGTH, CONCAT
2 GNB, WNB, CLOSEDEL, OPENX, BLOCKREAD, BLOCK WRITE
3 CLOSE, OPEN, PURGE, CHAIN, CREATE
4 WRD, HI, LO, SWAP, ADDR, SIZEOF, INLINE, EXIT, PACK, UNPACK
5 IORESULT, PAGE, NEW, DISPOSE
6 SUCC, PRED, EOF, EOLN
7 TSTBIT, CLRBIT, SETBIT, SHR, SHL
8 RESET, REWRITE, GET, PUT, ASSIGN, MOVELEFT, MOVE-RIGHT, FILL
9 READ, READLN
10 WRITE, WRITELN
11 nefolosit

12 MAXAVAIL, MEMAVAIL
13 SEEKREAD, SEEKWRITE
14 RIM85, SIM85, WAIT
15 READHEX, WRITEHEX

9.1.2. Utilizarea linkeditorului LINKMT

Sintaxa generală a comenzii de linkeditare este următoarea:
LINKMT [prog=] prog,modul,modul,...,PASLIB/opțiuni

Observații:

— se poate da, opțional, un nume fișierului de ieșire (de tip COM) separat prin "=" de numele fișierelor de intrare. În lipsa unui nume explicit, fișierul de ieșire primește numele primului fișier de intrare, care trebuie să fie programul principal.

— fișierele de intrare notate cu „modul" pot fi biblioteci de module relocabile sau module individuale, rezultate din compilare sau din asamblare cu M80, dar numai de tipul .ERL sau .CMD.

— sînt admise o serie de opțiuni de linkeditare, precedate de caracterul „/"; aceste opțiuni sînt de două tipuri:

— opțiuni individuale la fiecare fișier: /S,/F,/O;

— opțiuni globale, scrise la sfîrșitul liniei: /L,/M,/E,/P,/D.

Lista opțiunilor permise la linkeditare:

/L	listare adresă de cod și de date la fiecare modul
/E	listare nume puncte de intrare, inclusiv din biblioteci
/M	listare puncte de intrare sub formă tabelară
/P:nnnn	stabilește adresa „nnnn" pentru secțiunea de cod
/D:nnnn	stabilește adresa „nnnn" pentru secțiunea de date
/W	se scrie un fișier de tipul .SYM utilizabil de către SID
/F	fișierul anterior este un fișier de tip .CMD
/S	fișierul anterior este o bibliotecă căutată selectiv
/O:n	atribuie numărul „n" unui segment (Overlay)
/Vm:nnnn	stabilește adresa „nnnn" pentru segmentul „m"
/X:nnnn	stabilește memoria de date suplimentară la un segment

Fișiere de comenzi pentru linkeditor

Linkeditorul LINKMT poate citi numele fișierelor prelucrate fie de la consolă, fie dintr-un fișier de comenzi de tip .CMD. Fișierul de comenzi CMD conține o listă de nume de fișiere scrise pe linii separate sau pe o singură linie, dar separate prin virgule. Numărul maxim de caractere admise este de 256. Fișierul de comenzi CMD trebuie urmat de opțiunea /F. Exemplu:
În locul comenzii:

LINKMT CALC,TRANCEND,FPREALS,PASLIB/S

se poate introduce de la consolă comanda următoare:

LINKMT CALC/F

dacă a fost creat în prealabil următorul fişier CALC.CMD:

CALC
TRANCEND,FPREALS
PASLIB/S

Această soluţie este preferabilă ca timp de răspuns faţă de utilizarea unui fişier de comenzi de tip .SUB şi a programului SUBMIT din CP/M.

Relaţia dintre LINKMT şi L80

Formatul relocabil .ERL acceptat de LINKMT nu este identic, dar este compatibil cu formatul relocabil .REL acceptat de L80. Formatul ERL conţine majoritatea inregistrărilor din formatul REL, cu excepţia cîtorva înregistrări, corespunzătoare blocurilor de comun (COMMON), iniţializării de date în secţiunea de date şi cererilor de căutare în bibliotecă. Spre deosebire de formatul REL, în formatul ERL se impune o anumită ordine a înregistrărilor şi se acceptă nume publice de 7 caractere (faţă de 6 la REL).

LINKMT cere ca dimensiunea secţiunilor de program şi de date să preceadă primul octet de date care se încarcă; această condiţie este îndeplinită de către modulele relocabile generate de M80, dar nu şi de către modulele generate de compilatorul Fortran F80.

Consecinţele practice ale acestor diferenţe sînt următoarele:

— linkeditorul L80 nu acceptă formatul .ERL, dar este posibil să se transforme module ERL generate de Pascal/MT+ în module REL acceptate de L80, folosind bibliotecarul LIBMT;

— nu se pot lega împreună module Pascal cu module Fortran;

— modulele obiect generate de asamblorul M80 pot fi legate cu module Pascal folosind linkeditorul LINKMT, dacă sînt respectate următoarele condiţii:

— modulele generate de M80 au extensia .ERL şi nu .REL;

— modulele M80 nu folosesc directivele COMMON, .REQUEST şi nici directive DB, DW după DSEG.

Avantajele utilizării linkeditorului LINKMT faţă de L80:

— se pot lega programe mai mari decît cu L80;

— se pot genera programe segmentate;

— se poate genera şi format .HEX, pe lîngă formatul .COM;

— fişierul .COM este în general mai scurt, deoarece nu se scot pe disc zone de memorie neiniţializate.

9.1.3. Utilizarea depanatorului simbolic şi bibliotecarului

Depanatorul Pascal/MT+ se prezintă sub forma unui fişier relocabil, numit DEBUGGER.ERL, care se leagă împreună cu programul depanat astfel:

LINKMT prog = DEBUGGER,prog,PASLIB/S

unde „prog" este numele fișierului ce conține programul depanat, nume care se atribuie și fișierului de tip COM creat.

La execuție, controlul este preluat de către depanator.

Pentru o depanare simbolică este necesar ca la compilare să se utilizeze opțiunea $D în linia de comandă; această opțiune are ca efect generarea unui fișier de tip .PSY pe același disc cu fișierul .ERL. Linkeditorul LINKMT prelucrează fișierul .PSY și produce un fișier de tip .SYP, care conține numele simbolice din programul Pascal cu adresele de memorie asociate. Acest fișier este utilizat de către depanator, permițînd utilizarea de referințe simbolice în comenzile de depanare pentru variabile, proceduri și funcții (în locul unor adrese de memorie).

În cazul procedurilor (funcțiilor) recursive (compilate cu $S) nu se pot vizualiza variabilele locale, deoarece sînt alocate în stivă și nu au alocate adrese permanente.

Comenzile depanatorului au un cod mnemonic de două litere, urmat de un parametru. Parametrul poate fi:

— un nume de variabilă;

— un nume de variabilă precedat de un nume de procedură, separate prin caracterul ":";

— un număr zecimal sau un număr hexazecimal cu prefixul "$";

— un nume sau un număr urmate de caracterul „^" (semnul deadresare indirectă) indică conținutul unei adrese;

— o expresie formată dintr-un nume sau un număr urmat de operatorul „+" sau „—" și de un alt număr. Exemple

Fie următorul fragment de program Pascal:

```
TYPE PAOC = ARRAY [1..40] OF CHAR;
VAR ABC: INTEGER; PTR: ^PAOC;
```

Comenzile depanatorului pot utiliza următorii *parametri*:

ABC valoarea variabilei ABC (un număr întreg)

PTR valoarea variabilei PTR (o adresă)

PTR^tot tabloul de la adresa conținută în PTR

ABC+10 conținutul locației cu adresa ABC plus 10 octeți

PTR^+10 valoarea caracterului PAOC[11]

ABC—2 conținutul locației cu adresa ABC minus 2

$1A07 conținutul adresei absolute hexa 1A07H

$2C50^ 32 de octeți de la adresa conținută în locația 2C50H

$1FFD+$5B 32 de octeți de la adresa 2058H

$2C50^+10 32 de octeți de la adresa conținută în locația 2C50H adunată cu 10

PROC:I valoarea variabilei locale I din procedura PROC

PROC:P^+4 conținutul locației cu adresa calculată din conținutul variabilei P plus 4 din procedura PROC.

Lista comenzilor depanatorului Pascal

Comenzi de afişare a memoriei:

DVnume	afişează valoarea variabilei "nume"
DVnume^	afişează valorile de la adresa conţinută în variabila "nume"
DIpar	afişează un întreg
DCpar	afişează un caracter
DLpar	afişează un boolean (variabilă logică)
DRpar	afişează un real
DBpar	afişează un byte (un octet)
DWpar	afişează un cuvînt (Word)
DSpar	afişează un şir de caractere (String)
DXpar	afişează o structură pe lungime de 320 octeţi
DXpar,n	afişează o structură pe lungime de „n" octeţi
SEpar	afişează şi eventual modifică conţinutul unor locaţii de memorie; similară cu comanda "S" din DDT, se iese cu ".")

Comenzi cu execuţie controlată a programului depanat:

TR sau T	trasare: se execută o linie de program
Tn	trasare: se execută „n" linii de program
BE	executare program de la început (Begin Execution)
CO	continuare execuţie după un punct de oprire
SBnume	stabilire punct de oprire la începutul unei proceduri (Set Breakpoint)
RBnume	ştergere punct de oprire de la procedura "nume" (Remove Breakpoint)
E+	activează afişările la intrarea şi la ieşirea din fiecare procedură sau funcţie (implicit activ)
E—	dezactivează afişările la intrări (ieşiri) din proceduri
PN	afişează numele de proceduri din fişierul .SYP
VNnume	afişează numele variabilelor din procedura "nume"
??	afişează lista comenzilor depanatorului din fişierul DBUGHELP .TXT

Utilizarea bibliotecarului LIBMT

Bibliotecarul Pascal/MT+ are două funcţii:

— crearea de biblioteci prin concatenarea de fişiere .ERL;

— trecerea de la formatul .ERL la formatul .REL acceptat de L80.

Programul bibliotecar este apelat prin comanda:

LIBMT fişier

unde „fişier" este numele unui fişier de tip .BLD (tipul .BLD nu trebuie specificat explicit în comandă), creat cu un editor.

Fiecare linie din fişierul .BLD conţine un singur nume de fişier după cum urmează:

— numele fişierului de ieşire pe prima linie;
— numele fişierelor de intrare pe liniile următoare.

Dacă se doreşte crearea unui fişier de ieşire compatibil cu L80 atunci este necesar ca pe prima linie să se scrie L80, pe linia a doua numele fişierului de ieşire, ş.a.m.d.

Exemple de fişiere BLD:

1) Pentru utilizarea funcţiei de bibliotecar:

```
PLIB.ERL
PMOD1.ERL
PMOD2.ERL
```

S-a creat biblioteca PLIB din fişierele PMOD1, PMOD2.

2) Pentru utilizarea funcţiei de conversie format relocabil:

```
L80
PLIB.REL
PMOD1.ERL
PMOD2.ERL
```

Fişierele de intrare pot conţine module separate, rezultate din compilări (asamblări) sau alte biblioteci de module; tipul lor trebuie precizat, dar poate fi diferit de ERL. Bibliotecarul nu poate prelucra module compilate cu opţiunea $X.

Ordinea de introducere a modulelor în bibliotecă este importantă, deoarece linkeditorul LINKMT (ca şi L80) parcurge biblioteca o singură dată, extrăgînd numai modulele ce conţin simboluri referite anterior (cu opţiunea /S).

Dacă există o referire la un punct de intrare dintr-un modul, atunci LINKMT include tot modulul respectiv în programul creat, indiferent cîte proceduri fac parte din acel modul; de aceea este bine ca procedurile destinate introducerii în biblioteci să fie compilate ca module distincte.

Nu se recomandă modificarea bibliotecii PASLIB

Utilizarea dezasamblorului PASCAL

Dezasamblorul DIS8080 trebuie să primească ca intrări un fişier .PRN (rezultat dintr-o compilare cu opţiunea $P) şi un fişier .ERL extins (rezultat din folosirea opţiunii $X la compilare); el produce un fişier de ieşire conţinînd codul simbolic 8080 generat de compilator pentru fiecare instrucţiune Pascal, adresele variabilelor Pascal (ca adrese relative în modul) precum şi alte informaţii utile pentru depanare la nivel de instrucţiuni maşină.

Fişierul de ieşire poate fi scos pe disc sau pe orice alt dispozitiv CP/M (CON:, LST:); dacă nu se precizează ieşirea, atunci se listează la consolă.

Numărul de linii afişate pe o pagină este de 66, dar poate fi modificat printr-o opţiune (,L = nn) asociată fişierului de ieşire în linia de comandă. Exemplu:

PASCAL PROG $P@X
DIS8080 PROG,LST:,L = 58 sau DIS8080 PROG,PROG.DIS

9.2. Particularităţi de implementare Pascal MT+

9.2.1. Particularităţi ale limbajului Pascal-80

Limbajul Pascal/MT+ reprezintă o extindere a limbajului Pascal definit în standardul ISO şi în raportul PASCAL (Jensen, Wirth).

Principalele *facilităţi din standardul Pascal-ISO* prezente şi în Pascal/MT+ sînt:

— tipuri de date predefinite: REAL, INTEGER, CHAR, BOOLEAN,
ARRAY (tablouri cu mai multe dimensiuni),
RECORD (înregistrări, inclusiv înregistrări cu variante),
referinţă (Pointer) către date de orice tip,
FILE (fişiere de orice tip şi fişiere TEXT),
SET (mulţimi cu maxim 256 de elemente).

— tipuri enumerate, subdomeniu şi tipuri definite de utilizator;

— proceduri şi funcţii, care pot avea ca parametri nume de proceduri sau funcţii;

— tablouri conforme ca parametri de proceduri: o procedură poate avea ca argument tablouri de dimensiuni diferite, dar cu elemente de acelaşi tip şi cu indici de acelaşi tip (în acelaşi subdomeniu de valori);

— fişiere standard INPUT şi OUTPUT, ca parametri de program;

— toate procedurile standard referitoare la fişiere în forma lor originală: RESET, REWRITE, GET, PUT, READ, WRITE, READLN, WRITELN;

— fişiere de orice tip şi fişiere TEXT;

— toate instrucţiunile standard Pascal, inclusiv WITH;

— salt în afara unei proceduri (nerecursive);

— atributele PACK şi UNPACK, dar fără nici un efect, deoarece toate structurile sînt împachetate la nivel de octet;

— procedurile standard NEW şi DISPOSE de gestiune dinamică a memoriei, cu reutilizarea memoriei eliberate (memorie HEAP);

— comentariile pot fi încadrate de "{" şi "}" sau de "(*" şi "*)"

Extensii Pascal/MT+ faţă de standard

1) Un nou caracter alfabetic: caracterul "@", folosit în numele unor rutine de bibliotecă. Deoarece standardul ISO defineşte caracterul "@" ca echivalent pentru caracterul " ↑ " (indicator de referenţiere sau

de adresare indirectă) este prevăzută opţiunea de compilare "@", care exclude acest caracter din mulţimea literelor şi îi atribuie semnificaţia din standard.

2) Identificatorii (numele simbolice) pot avea orice lungime, dar numai primele 8 caractere sînt semnificative (celelalte fiind ignorate de compilator). În interiorul unui nume poate fi utilizat şi caracterul special de subliniere.

3) Sînt permise constante hexazecimale, precedate de caracterul $. Cifrele hexa A,B,C,D,E,F, pot fi scrise şi ca a,b,c,d,e,f.

4) S-a adăugat o nouă constantă de tipul şir: şirul nul (' ').

5) Noi tipuri de date predefinite:

BYTE: definit ca subdomeniu 0..255, cu proprietatea că este compatibil cu tipul CHAR.

WORD: definit ca întreg fără semn (subdomeniul 0..32767)

STRING: tablou împachetat de caractere, care conţine în primul octet lungimea efectivă a şirului; compatibil cu tipul CHAR. Lungimea maximă implicită a unui şir (STRING) este de 80, dar la declararea unei variabile STRING se poate specifica lungimea maximă a şirului (max 255), ca în exemplul următor:

```
VAR NUME: STRING [25]
```

6) Atributul EXTERNAL la declararea de variabile externe, introdus înaintea numelui variabilei:

```
EXTERNAL COMM: INTEGER;
```

7) Atributul ABSOLUTE la declaraţii de variabile, cu sintaxa:

```
VAR nume: ABSOLUTE [const] tip
```

unde „const" este o constantă (de obicei în hexa), care specifică adresa absolută de memorie a variabilei declarate. Exemplu:

```
VAR BDOS: ABSOLUTE [6] WORD;
ECRAN: ABSOLUTE [$C000] ARRAY [0..23] OF ARRAY
[0..79] OF CHAR;
```

Nu sînt permise variabile absolute de tipul STRING la adrese mai mici de 100H (datorită modului de lucru al compilatorului)

8) Operatori logici pentru variabile nebooleene, cu lungimea de unul sau de doi octeţi (INTEGER, WORD, BYTE, CHAR):

& pentru "şi" logic;

! sau | pentru "sau" logic;

~ sau ? pentru "nu" (complement faţă de 1).

Aceşti operatori au aceeaşi prioritate ca operatorii analogi pentru variabile booleene.

9) Instrucţiunea de atribuire admite următoarele atribuiri între tipuri diferite:

— unei variabile de tip BYTE i se poate atribui o variabilă de tip CHAR sau o constantă de tip caracter între ghilimele;

— unei variabile de tip STRING i se poate atribui rezultatul unei expresii de tip CHAR.

10) Instrucţiunea CASE poate avea o clauză ELSE, care specifică ce acţiune are loc în cazul în care expresia de selecţie este diferită de toţi selectorii de caz.

11) Un nou tip de procedură: procedura de tratare a unei întreruperi, definită astfel:

PROCEDURE INTERRUPT [n] nume;

unde "n" este nivelul de întrerupere tratat de procedură.
Procedurile de tip INTERRUPT pot fi definite numai în programul principal şi nu pot avea parametri.

12) Atributul EXTERNAL înaintea antetului de procedură (înaintea cuvîntului PROCEDURE) pentru proceduri externe, compilate separat sau scrise în limbaj de asamblare. Procedurile externe trebuie declarate înainte de prima procedură internă (locală).

13) Variabilele de tip fişier (FILE) sînt asociate automat de către compilator cu fişiere disc temporare, avînd numele PASTMPnn şi extensia .$$$ (nn = 00,01,02,..). Un asemenea fişier este creat la apelarea procedurii REWRITE. În standardul Pasçal variabilele fişier au asociate automat cîte o zonă de memorie internă.

14) O nouă funcţie standard pentru conversie de tip: WRD(x) transformă o variabilă sau o expresie „x" de tip întreg în tipul WORD (întreg fără semn).

15) Noi proceduri şi funcţii standard pentru:
— operaţii pe şiruri de caractere;
— operaţii la nivel de octet şi de bit;
— operaţii cu fişiere secvenţiale şi în acces direct;
— diverse alte funcţii şi proceduri.
Aceste proceduri vor fi prezentate în detaliu mai departe.

16) Cuvîntul cheie INLINE permite inserarea de instrucţiuni maşină simbolice sau numerice într-un text Pascal, după sintaxa următoare:

INLINE (arg/arg/arg/...);

unde fiecare dintre argumentele notate cu „arg" poate fi:
— un cod numeric de instrucţiune 8080 sau Z80, ca un caz particular de constantă Pascal (de obicei în hexazecimal);
— un cod mnemonic de instrucţiune 8080, precedat de un apostrof dublu şi în care spaţiile şi virgula sînt ignorate:

"MOV A,M/ sau "MOVAM/

— orice constantă Pascal;
— orice variabilă scalară din programul Pascal;
— o expresie de adresare autorelativă de forma

$* + n$ sau $* - n$

unde „*" desemnează adresa instrucţiunii curente.

Exemplu de secvenţă INLINE:

INLINE ("IN/$80/"ANI/$02/"JNZ/* — 4);

17) Efectul procedurilor NEW şi DISPOSE depinde de modul cum a fost linkeditat programul — cu sau fără fişierul FULLHEAP. ERL.

Dacă nu se foloseşte FULLHEAP, atunci biblioteca PASLIB conţine două rutine de gestiune a memoriei dinamice, organizată ca stivă cu efectul următor:

NEW pune în stivă variabila dinamică adresată de parametru

DISPOSE nu are nici un efect

Stiva pentru variabile dinamice creşte de la sfîrşitul zonei de date (statice) pînă la stiva hardware (adresată de registrul SP).

Dacă se foloseşte fişierul FULLHEAP, atunci memoria dinamică este organizată ca „Heap", iar efectul procedurilor este:

NEW caută cea mai mică zonă de memorie suficientă pentru variabila dinamică specificată şi alocă această zonă;

DISPOSE eliberează zona de memorie ocupată de o variabilă dinamică şi o întoarce în lista spaţiului disponibil.

Compactarea zonelor libere, adiacente din lista spaţiului disponibil se face fie de procedura NEW, fie de procedura MAXAVAIL care înapoiază lungimea celei mai mari zone de memorie disponibile

9.2.2. Realizarea de programe modulare în Pascal MT+

Compilări separate (modulare)

Unitatea de compilare în Pascal/MT+ este fie programul, fie modulul Pascal. Un modul are o structură asemănătoare unui program;

— începe cu o instrucţiune specială: MODULE nume;

— poate avea o parte de declaraţii pentru etichete, constante, tipuri de date şi variabile;

— conţine declaraţii de proceduri şi (sau) de funcţii;

— se termină prin cuvîntul cheie MODEND, urmat de un punct.

Spre deosebire de un program standard Pascal, nu este necesară într-un modul o secvenţă de instrucţiuni între BEGIN şi END.

Variabilele declarate la începutul unui modul pot fi tratate ca variabile locale sau ca variabile globale (externe), în funcţie de opţiunea de compilare $E:

{$E—} înseamnă că variabilele sînt considerate locale

{$E+} înseamnă că variabilele sînt considerate globale

Prin aceste variabile se poate comunica între module separate.

Pentru utilizarea de variabile sau funcţii compilate în alte module decît cel care le apelează, se utilizează atributul extern (EXTERNAL) în declaraţiile de variabile sau de proceduri:

1) VAR I, J, K : EXTERNAL INTEGER;
CX: EXTERNAL RECORD RE, IM: REAL END;

2) EXTERNAL PROCEDURE SORT(VAR Q: LIST; LEN: INTEGER);

3) EXTERNAL FUNCTION IOTEST: INTEGER;

Declarațiile de variabile sau proceduri externe pot apărea numai la nivelul exterior dintr-un program (modul).

Concordanța declarațiilor externe din module diferite referitoare la aceleași variabile, funcții (proceduri) și parametri de proceduri cade în sarcina programatorului, deoarece nu poate fi verificată nici de compilator, nici de linkeditor.

Procedurile scrise în limbaj de asamblare și apelate din Pascal trebuie de asemenea declarate ca proceduri externe.

Numele de simboluri globale Pascal/MT+ sînt limitate la 7 caractere semnificative sau 6 caractere pentru limbaj de asamblare.

Înlănțuirea de programe PASCAL („Chaining")

Înlănțuirea de programe poate constitui o alternativă la segmentare pentru programe mari.

Un program poate încărca de pe disc un alt program și să-i predea controlul după cum urmează:

— declară o variabilă fișier fără tip (FILE;) și asociază această variabilă cu numele extern al fișierului ce conține un alt program folosind procedura ASSIGN;

— inițializează (deschide) fișierul program cu procedura RESET;

— încarcă și lansează programul încărcat, folosind procedura CHAIN cu numele variabilei fișier ca argument.

Transmiterea de date între două programe înlănțuite se face:

— prin variabile globale, cu același nume și plasate la aceleași adrese (cu opțiunea /D la linkeditare);

— prin variabile absolute de memorie, cu adrese identice în cele două programe.

În ambele cazuri, programatorul trebuie să se asigure că programul înlănțuit nu se încarcă peste datele comune, mai ales atunci cînd un program scurt se înlănțuie cu un program lung.

Pentru conservarea memoriei dinamice la o înlănțuire, trebuie ca variabila SYSMEM (care indică începutul memoriei dinamice) să fie declarată ca EXTERNAL INTEGER.

Segmentarea programelor Pascal

Un program mare, care nu încape în întregime în memorie poate fi segmentat într-un segment rădăcină, rezident în memorie, și mai multe segmente de suprapunere („Overlays"), încărcate succesiv la aceleași adrese de memorie.

Încărcarea şi reîncărcarea segmentelor nerezidente nu trebuie programată explicit de către utilizator, deoarece este prevăzut în biblioteca PASLIB un modul special de gestiune a segmentării. Acest modul verifică, la apelarea unei proceduri dintr-un segment nerezident, dacă segmentul respectiv este în memorie, iar în caz contrar asigură încărcarea de pe disc a acestui segment.

Programatorul are sarcina să definească conţinutul fiecărui segment, numărul segmentului, adresa de încărcare în memorie şi alte informaţii necesare pentru gestiunea automată a segmentelor.

Un program segmentat ocupă în memorie mai multe zone adiacente:

— zona segmentului rădăcină („Root area"), la adrese mici;

— zona sau zonele de suprapunere („Overlay area"), unde se încarcă segmentele paralele nerezidente;

— zona de date a segmentului rădăcină (precizată prin /D);

— zona de date a segmentelor nerezidente;

— zona de alocare dinamică, utilizată prin NEW şi DISPOSE

În cazul cel mai simplu există o singură zonă de încărcare a segmentelor nerezidente, dimensionată în funcţie de segmentul cel mai mare dintre segmentele paralele aduse în zona respectivă. Pot exista 16 zone de suprapunere diferite, succesive.

Numărul de segmente paralele, asociate cu o zonă de suprapunere, este de cel mult 16 segmente.

Un segment poate conţine unul sau mai multe module compilate, separat de modulele din alte segmente, şi constituie un fişier.

Încărcarea unui segment nerezident se face în urma apelării unei proceduri conţinute în segmentul respectiv, apel care poate proveni din rădăcină sau dintr-un alt segment, inclusiv din segmente paralele.

La declararea procedurilor externe din alte segmente trebuie precizat şi numărul segmentului conform sintaxei:

EXTERNAL n PROCEDURE ...

sau

EXTERNAL n FUNCTION ...

unde „n" este numărul segmentului ce conţine procedura (funcţia).

Segmentele unui program se linkeditează separat şi trebuie să aibă toate acelaşi nume de fişier, dar cu extensii diferite: segmentul rădăcină are extensia .COM, iar celelalte segmente trebuie să aibă tipul .001, .002, .003, .. Nu este necesară numerotarea în continuare a segmentelor, dar segmentele 1—16 sînt încărcate în zona de suprapunere 1, segmentele 17—32 în zona 2 ş.a.m.d.

Înainte de linkeditarea finală a segmentelor este necesară o linkeditare pentru determinarea lungimii secţiunilor de cod şi de date ale fiecărui segment.

Comanda de linkeditare a rădăcinii are forma următoare:

LINKMT prog, ⟨alte module⟩, PASLIB/S/Vn: mmmm/D: dddd/X: pppp

unde:

„prog" este numele fişierului rădăcină (şi numele segmentului);
„n" este numărul zonei de suprapunere (în hexazecimal);
„mmmm" este adresa de memorie a zonei n (în hexa);
„dddd" este adresa secţiunii de date (în hexa) şi poate lipsi;
„pppp" este dimensiunea memoriei necesare pentru datele locale din segmentele nerezidente şi poate lipsi.

Comanda de linkeditare a unui segment nerezident are forma:

LINKMT prog = prog/O: n, ⟨alte module⟩, PASLIB/S/P: mmmm

Este necesar ca linkeditarea rădăcinii să se facă înainte de celelalte segmente, deoarece opţiunea /V are drept efect generarea unui fişier „prog.SYM", care conţine numele simbolice globale, fişier care este necesar la linkeditarea celorlalte segmente; opţiunea /O: n se aplică asupra fişierului de tipul .SYM şi nu asupra unui fişier de tipul .ERL.

Adresa zonei sau zonelor de suprapunere se poate determina printr-o linkeditare prealabilă a fişierului rădăcină, fără opţiunea /V, şi notarea lungimilor secţiunii de cod şi de date.

Dacă nu s-a folosit opţiunea /D, atunci adresa primei zone de suprapunere este egală sau mai mare cu 100H + cod + date (100H este adresa de încărcare a întregului program). Dacă s-a folosit opţiunea /D pentru a plasa secţiunea de date la altă adresă, atunci adresa zonei de suprapunere este cel puţin egală cu 100H + cod, unde „cod" este lungimea secţiunii de cod a rădăcinii.

Adresa astfel calculată apare atît la opţiunea /V pentru rădăcină, cît şi la opţiunea /P de la fiecare segment suprapus.

Opţiunea /X este necesară numai atunci cînd programul segmentat foloseşte procedurile standard de gestiune dinamică a memoriei NEW, DISPOSE. Ea specifică memoria necesară pentru datele locale din segmentele suprapuse şi permite linkeditorului să determine adresa de început a memoriei dinamice. Dimensiunea acestei zone este egală cu suma zonelor de date din segmentele încărcate la aceeaşi adresă.

Este posibil ca unele segmente sau toate să se afle pe alte unităţi de disc decît unitatea de unde s-a încărcat rădăcina; în acest caz se va folosi procedura @OVS pentru precizarea unităţii pe care se află un segment. Procedura @OVS se va declara astfel:

EXTERNAL PROCEDURE @OVS (n: INTEGER; d: CHAR);

unde: „n" este numărul segmentului
„d" este numele unităţii de disc (@, A, B,..)

În mod normal nu este permisă apelarea unui segment S2 dintr-un

alt segment paralel S1, deoarece nu este asigurată reîncărcarea automată a lui S1 după terminarea lui S2 şi deci nu se face revenirea corectă din S2 în S1 (această reîncărcare consumă timp şi ar fi inutilă în majoritatea cazurilor). Pentru a fi posibilă apelarea unui segment din alt segment paralel trebuie legată la programul principal o altă variantă a modulului de gestiune a segmentării, aflată în fişierul ROVLMGR.ERL, în locul celui din PASLIB.

Linkeditarea unui program segmentat trebuie refăcută numai dacă s-au efectuat modificări în rădăcină; dacă s-a modificat un segment, atunci este suficientă linkeditarea segmentului respectiv.

9.2.3. Utilizarea de subrutire în limbaj maşină

Editorul de legături LINKMT poate lega module obiect provenite din compilare PASCAL (MT+) cu module obiect rezultate din asamblare cu M80, dacă sînt îndeplinite următoarele condiţii:

— simbolurile globale nu depăşesc 6 caractere în lungime;

— nu se utilizează caracterul „$" în nume simbolice globale;

— nu se utilizează directiva COMMON în limbajul de asamblare;

— nu se utilizează directive DB, DW în secţiunea de date (DSEG).

Comunicarea de date între Pascal şi asamblor se poate face:

— prin variabile globale Pascal;

— prin parametri de proceduri sau prin funcţii.

Variabilele globale Pascal (declarate în programul principal) pot fi utilizate din limbaj de asamblare, dacă sînt declarate ca simboluri globale (cu directiva EXTRN sau PUBLIC) şi dacă s-a utilizat opţiunea {$E +} la compilarea Pascal.

Pentru a utiliza corect conţinutul variabilelor Pascal, trebuie să se cunoască modul de alocare a adreselor de memorie pentru fiecare tip de variabilă Pascal.

Variabilele globale Pascal sînt plasate în memorie la adrese succesive, în ordinea declarării lor, astfel:

— cîte un octet pentru variabile de tip CHAR, BYTE, BOOLEAN;

— cîte doi octeţi pentru variabile de tip INTEGER, WORD: întîi octetul inferior şi apoi octetul superior;

— cîte 4 octeţi pentru variabile de tip REAL (binar virgulă mobilă);

— cîte 32 de octeţi pentru variabile de tip SET;

— între 1 şi 256 octeţi pentru variabile de tip STRING; primul octet conţine lungimea efectivă a şirului.

Componentele unui tablou cu două sau mai multe dimensiuni sînt memorate pe linii: linia 1, linia 2 etc.

Parametrii procedurilor şi funcţiilor se transmit prin stivă:

— adresa de revenire din procedură în vîrful stivei, pe 2 octeţi;

— valoarea sau adresa parametrilor, în ordine inversă: ultimul parametru se află imediat după adresa de revenire.

Pentru fiecare parametru se introduc în stivă unul sau mai multe cuvinte de cîte 2 octeți, inclusiv pentru variabile cu lungime de un octet; pentru parametrii scalari, care nu sînt precedați de VAR, se transmit valorile parametrilor; pentru parametrii nescalari și pentru parametrii scalari precedați de VAR se pun în stivă adresele parametrilor. Exemplu:

PROCEDURE ASM (I, J: INTEGER; VAR Q: STRING; C, D: CHAR);

La intrarea în procedură ASM stiva are următorul conținut:

(SP) : adresa de revenire
(SP+2) : D, 0
(SP+4) : C, 0
(SP+6) : adresa lui Q
(SP+8) : J (octet inferior, octet superior)
(SP+10) : I (octet inferior, octet superior)

Subrutina în limbaj de asamblare trebuie să scoată din stivă toți parametrii înainte de revenirea în programul apelant.

Rezultatul unei funcții (ca valoare) rămîne în vîrful stivei după revenirea din subprogramul funcție:

— se scoate adresa de revenire din stivă și se păstrează;
— se scot toți parametrii din stivă;
— se pune rezultatul funcției în stivă;
— se pune înapoi în stivă adresa de revenire.

În limbaj de asamblare sînt permise numai funcții cu rezultat scalar de tipul INTEGER, REAL, BOOLEAN, CHAR.

9.3. Proceduri și funcții predefinite în Pascal/MT+

9.3.1. Prelucrări pe șiruri de caractere și pe biți

Proceduri de lucru cu tablouri de caractere:

procedura *MOVE* (s, d, n);
procedura *MOVELEFT* (s, d, n);
procedura *MOVERIGHT* (s, d, n);

unde: „s" = sursa; „d" = destinația; „n" = număr de caractere
Procedurile MOVE și MOVELEFT sînt echivalente.
MOVELEFT mută „n" caractere de la stînga la dreapta.
MOVERIGHT mută „n" caractere de la dreapta la stînga
Variabilele sursa „s" și destinație „d" pot fi de orice tip, inclusiv referințe (pointeri) sau întregi utilizați ca pointeri.
Sursa și destinația pot fi de tipuri diferite. Parametrul „n" poate fi orice expresie întreagă, cu rezultat între 0 și 65535.

procedura *FILLCHAR* (d, n, c);

unde: „d" = destinaţie; „n" = număr de caractere; „c" = caracter. Umple tabloul destinaţie „d" cu „n" caractere de valoare „c". Destinaţia „d" este un tablou de caractere; „n" este orice expresie cu rezultat întreg; „c" este o variabilă de tip CHAR sau un literal (un caracter între ghilimele).

Funcţii şi proceduri de prelucrare şiruri de caractere:

funcţia *CONCAT* (s1, s2,...,sn) de tip STRING
unde: s1, s2,..sn sînt fie variabile de tip STRING, fie constante de tip şir sau de tip caracter. Are ca rezultat un şir de caractere obţinut prin concatenarea şirurilor sursă „s1", „s2",... şi trunchiat la primele 256 de caractere.

funcţia *COPY* (s, d, n) de tip STRING
unde: „s" = sursa; „d" = destinaţia; „n" = număr de caractere. Copiază „n" caractere din şirul sursă „s" la adresa destinaţie „d". Sursa „s" este un şir (STRING) iar parametrii „d" şi „n" sînt expresii cu rezultat întreg.

Funcţiile CONCAT şi COPY au un singur buffer pentru şirul rezultat, de aceea în urma unor apeluri succesive ale funcţiei rămîne în buffer numai rezultatul ultimei execuţii.

procedura *DELETE* (s, i, n)
unde: „s" = un şir de caractere; „i" şi „n" = expresii întregi. Elimină „n" caractere începînd din poziţia „i" a şirului „s".

procedura *INSERT* (s, d, i)
unde: „s" = şir sau caracter sursă, „d" = şir destinaţie; „i" = index în şirul destinaţie (o expresie întreagă). Inserează caracterul sau caracterele sursă „s" în poziţia „i" din şirul destinaţie „d".

funcţia *LENGTH*(s) de tip INTEGER
unde „s" = variabilă sau constantă de tip STRING.
Are ca rezultat lungimea unui şir de caractere.

funcţia *POS* (p, s) de tip INTEGER
unde „p" = un caracter sau un şir de caractere (variabilă sau constantă); „s" = un şir de caractere sursă. Are ca rezultat poziţia primei apariţii a şirului sau caracterului „p" în şirul „s".

Funcţii şi proceduri la nivel de octet şi de bit. Toate aceste rutine operează numai asupra unei variabile de unul sau de doi octeţi, deci de tipul CHAR, BYTE, BOOLEAN, WORD sau INTEGER, numită în continuare „variabila de bază".

procedura *CLRBIT* (v, b)
unde: „v" = variabila de bază; „b" = număr de bit (0..15). Şterge (pune pe 0) bitul numărul „b" din variabila „v". Bitul 0 este în dreapta şi bitul 15 la stînga.

procedura *SETBIT* (v, b)
Pune pe 1 bitul numărul „b" din variabila de bază „v".

funcţia *TSTBIT* (v,b) de tip BOOLEAN

Are un rezultat TRUE (adevărat) dacă bitul numărul „b" din variabila de bază „v" are valoarea 1 și un rezultat FALSE dacă acest bit este 0.

funcția *HI* (v) de tip INTEGER

Are ca rezultat întreg octetul superior (High) din variabila de bază „v".

funcția *LO* (v) de tip INTEGER

Are ca rezultat întreg octetul inferior (Low) din variabila de bază „v".

funcția *SWAP* (v) de tip INTEGER

Schimbă între ei octetul inferior și octetul superior al variabilei de bază „v"; are rezultat de tip întreg.

funcția *SHR* (v, m) de tip INTEGER

unde: „v" = variabila de bază; „m" = expresie cu rezultat întreg. Deplasează la dreapta cu „m" biți conținutul variabilei „v" introducînd zerouri prin stînga („v" de 8 sau de 16 biți).

funcția *SHL* (v, m) de tip INTEGER

unde: „v" = variabila de bază; „m" = expresie cu rezultat întreg. Deplasează la stînga cu „m" biți conținutul variabilei „v", introducînd zerouri prin dreapta („v" de 8 sau de 16 biți).

9.3.2. Funcții și proceduri diverse

procedura *EXIT*:

Termină execuția unui program, proceduri sau o funcție Pascal.

funcția ADDR (id) de tip INTEGER;

Are ca rezultat adresa asociată identificatorului „id", care poate fi un nume de variabilă, de procedură sau de funcție, de variabilă indexată sau de componentă a unei înregistrări; se poate folosi și pentru nume externe și din alte segmente. Numele „id" trebuie să fie „vizibil" din locul unde este folosită ADDR; de exemplu un nume local dintr-o procedură nu este vizibil din programul principal sau din altă procedură de același nivel.

funcția *SIZEOF* (id) de tip INTEGER

unde: „id" este numele unei variabile sau unui tip de date. Are ca rezultat numărul de octeți necesar pentru o variabilă sau pentru un tip de date definit de utilizator.

funcția *MEMAVAIL* de tip INTEGER

Are ca rezultat dimensiunea celui mai mare bloc de memorie dinamică neutilizat de procedurile NEW și DISPOSE.

funcția *MAXAVAIL* de tip INTEGER

Colectează resturile de memorie eliberate prin DISPOSE, combină zonele libere adiacente și întoarce lungimea celui mai mare bloc de memorie disponibil pentru alocare dinamică.

procedura *WAIT* (port, masca, sens)

unde „port" și „masca" sînt constante cu nume sau literale; „sens" este o constantă logică (TRUE/FALSE).

Este echivalentă cu următoarea secvență de instrucțiuni:

IN port
ANI masca
J?? $-4

unde ?? este Z dacă sens = FALSE și ?? = NZ dacă sens = TRUE.

funcția *@CMD* de tip ^STRING

Are ca rezultat adresa unui șir de caractere, care conține coada liniei de comandă prin care a fost lansat programul (deci parametrii comenzii). Acest șir începe întotdeauna cu un blanc, care separă numele comenzii de coada comenzii. Această funcție poate fi folosită o singură dată, la începutul programului, înainte de a deschide un fișier.

9.3.3. Proceduri și funcții pentru fișiere

funcția *IORESULT* de tip INTEGER;

În urma operațiilor de I/E cu fișiere, funcția întreagă fără parametrii IORESULT întoarce octetul de rezultat furnizat de BDOS. După deschiderea sau închiderea unui fișier IORESULT = 255 arată o eroare la aceste operații (terminare anormală). După o citire sau o scriere, o valoare nenulă pentru IORESULT arată o terminare anormală a operației: sfirșit de fișier la o citire secvențială (READ, REALN, GET) sau eroare de citire (scriere) în acces direct sau secvențial. Unele proceduri (OPEN, CLOSE,..) au și un parametru pentru rezultatul operației, parametru care primește aceeași valoare ca și IORESULT.

procedura *ASSIGN* (f, nume);

unde: „f" = variabilă de tip FILE; „nume" = nume de fișier disc sau de dispozitiv periferic CP/M (variabilă sau constantă șir de caractere). Asociază o variabilă fișier Pascal cu un nume extern de fișier sau cu un nume de dispozitiv logic de I/E. Fișierul „f" poate avea componente de orice tip, dar dacă se asociază cu un nume de dispozitiv, atunci variabila „f" trebuie să fie de tipul TEXT. Numele de dispozitive admise sînt:

CON: consola la intrare și la ieșire;
KBD: claviatura consolei (numai intrare) (incompatibilă cu CON:);
TRM: ecranul consolei (numai ieșire);
LST: imprimanta;
RDR: dispozitiv de citire;
PUN: dispozitiv de perforare (de scriere).

Citirea de la KBD și afișarea la TRM nu interpretează nici un caracter; citirea de la CON interpretează caracterele CR (înlocuit cu CR, LF) și BS (înlocuit cu BS, SP, BS). Afișarea la CON: interpretează caracterele CR (ca CR, LF) și TAB (salt la 8 spații). La procedura ASSIGN trebuie să se facă apel înainte de a deschide un fișier prin procedurile RESET

REWRITE; în caz contrar se utilizează numele implicite PASTMPnn. $$$ pentru fişiere.

procedura *OPEN* (f, nume, r);

unde: „f" = variabilă de tip FILE; „nume" = şir de caractere; „v" = variabilă întreagă.

Deschide fişierul desemnat prin „f" şi asociază această variabilă cu fişierul extern specificat prin „nume"; variabila „r" conţine rezultatul deschiderii (255 = eroare). OPEN (f, nume, r) are acelaşi efect cu secvenţa: ASSIGN (f, nume); RESET (f); r: = IORESULT;

procedura *CLOSE* (f, r);

unde: „f" = variabilă de tip FILE; „r" = variabilă de tip INTEGER. Închide fişierul desemnat prin variabila „f" şi atribuie variabilei „r" rezultatul operaţiei (255 = eroare).

procedura *CLOSEDEL* (f, r);

Închide şi apoi şterge fişierul desemnat prin variabila „f"; variabila întreagă „r" primeşte rezultatul (255 = eroare).

procedura *PURGE* (f);

Şterge fişierul desemnat prin variabila fişier „f", asociată anterior cu un nume de fişier prin ASSIGN.

funcţia *GNB* (f) de tip CHAR;

Citeşte octetul următor din fişierul desemnat prin variabila „f" de tipul FILE OF ARRAY OF CHAR (Get Next Byte).

funcţia *WNB* (f, c) de tip BOOLEAN

Scrie caracterul „c" în fişierul desemnat de variabila „f", de tip FILE OF ARRAY OF CHAR (Write Next Byte). Rezultatul lui WNB este TRUE dacă s-a produs o eroare la scriere.

Se recomandă ca dimensiunea tablourilor de caractere componente ale fişierelor exploatate prin WNB şi GNB să fie între 128 şi 4096 de octeţi; cu cît este mai mare acest buffer, cu atît timpul de prelucrare a fişierului este mai mic. Viteza sporită de lucru reprezintă de altfel motivul utilizării funcţiilor WNB şi GNB în locul procedurilor standard PUT şi GET.

procedura *BLOCKREAD* (f, buf, r, 1, blk);
procedura *BLOCKWRITE* (f, buf, r, 1, blk);

unde: „f" = variabilă de tip FILE; „buf" = variabilă ARRAY; „r" = variabilă de tip INTEGER; „1" = expresie cu rezultat întreg; „blk" = expresie cu rezultat întreg.

BLOCKREAD citeşte din fişierul desemnat prin variabila „f" în tabloul „buf" (un buffer) blocul cu numărul „blk" în lungime de „1" octeţi; variabila „r" primeşte rezultatul operaţiei.

BLOCKWRITE scrie în fişierul „f" conţinutul tabloului „buf" în blocul cu numărul „blk" în lungime de „1" octeţi"; variabila „r" conţine rezultatul (eroare dacă „r" diferit de 0).

Dacă blk = —1, atunci se citeşte sau se scrie blocul următor din fişier (acces secvenţial). Valoarea maximă a lui „blk" este 32767. Dimen-

siunea zonei tampon „buf" trebuie să fie egală cu lungimea „l" a blocurilor din fişier şi ambele trebuie să fie multiplu de 128 (lungimea unei înregistrări disc).

procedura *SEEKREAD* (f, n);

Citeşte înregistrarea numărul „n" din fişierul asociat variabilei fişier „f" în variabila tampon „f^".

procedura *SEEKWRITE* (f, n);

Scrie din variabila tampon „f^" în înregistrarea cu numărul „n" din fişierul asociat variabilei fişier „f". Fişierele scrise cu SEEKWRITE sînt întotdeauna continue, indiferent de lungimea înregistrării, ceea ce înseamnă că aceste fişiere pot fi exploatate fie secvenţial fie în acces direct.

Rezultatul procedurilor SEEKxxxx se află în IORESULT.

Procedurile SEEKWRITE şi SEEKREAD nu se află în PASLIB; ele fac parte din fişierul RANDOMIO.ERL.

procedura *READHEX* (f, w, b);

procedura *WRITEHEX* (f, e, b);

unde: „f" = variabilă de tip TEXT; „w" = variabilă de orice tip; „e" = expresie de orice tip; „b" = întreg în 1..2

READHEX citeşte din fişierul asociat variabilei „f" în variabila „w" o valoare hexazecimală de „b" octeţi.

WRITEHEX scrie în fişierul asociat cu „f" valoarea expresiei „e" în hexazecimal, pe „b" octeţi.

Procedurile WRITEHEX, REAHEX pot fi folosite şi pentru afişarea sau citirea de numere hexazecimale la sau de la consolă sau la imprimantă.

9.4. Exemple de programe Pascal

1) Utilizare de tablouri conforme, cu limite variabile, la apelarea unei proceduri

```
PROGRAM CONFARRAY;
TYPE
 NATURAL=0..MAXINT;
VAR
 A1:ARRAY[1..10] OF INTEGER;
 A2:ARRAY[2..20] OF INTEGER;
PROCEDURE ARRLST (
  VAR AR1:ARRAY[MINI..MAXI:NATURAL] OF INTEGER );
 (* aici se definesc 3 variabile AR1,MINI,MAXI *)
VAR I:NATURAL;
BEGIN
 FOR I:=MINI TO MAXI DO
   WRITELN ('Array [',I,']=',AR1[I] )
 END;
```

```
BEGIN  (*program principal *)
  ARRLST (A1); (* limitele 1 si 10 se transmit implicit *)
  ARRLST (A2); (* limitele 2 si 20 se transmit implicit *)
END.
```

2) Exemplu de utilizare INLINE:

```
FUNCTON @NBDOS (FUNC:INTEGER; PARM:WORD):INTEGER;
 CONST CPM=5;
 VAR RESULT:INTEGER;
 BEGIN INLINE(
  "LHLD/FUNC/      {sau $2A/FUNC}
  "MOV C,L/        {sau $4D}
  "LHLD/PARM/      {sau $2A/PARM}
  "XCHG/           {sau $EB}
  "CALL/CPM/
  "MOV L,A/ MVI H,O/ SHLD/ RESULT);
 @BDOS:=RESULT;
END:
```

3) Procedură de inserare a unui şir sursă într-un şir destinaţie, folosind proceduri pe şiruri de caractere

```
PROCEDURE MOVESYTR;
CONST STRINGSZ=80;
VAR BUFFER:STRING [STRINGSZ];
    LINE:STRING;
PROCEDURE INSRT (VAR DEST:STRING; INDEX:INTEGER;
                 VAR SOURCE:STRING);
BEGIN
 IF LENGTH(SOURCE) <= STRINGSZ-LENGTH(DEST) THEN
   BEGIN
    MOVERIGHT(DEST[INDEX],DEST[INDEX+LENGTH(SOURCE)],
               LENGTH(DEST)-INDEX+1);
    MOVELEFT (SOURCE[1],DEST[INDEX],LENGTH(SOURCE));
    DEST[O]:=CHR(ORD(DEST[O])+LENGTH(SOURCE))
   END;
END;
```

4) Exemplu de utilizare proceduri la nivel de bit

```
PROCEDURE BITS;
VAR I:INTEGER;
BEGIN
 I:=O;
 SETBIT (I,5);
 IF I=32 THEN
  IF TSTBIT(I,5) THEN
    WRITELN ('I=',I);
 CLRBIT(I,5);
 IF I=O THEN
  IF NOT (TSTBIT (I,5)) THEN
    WRITELN ('I=',I);
END;
```

5) Procedură de afişare a unor adrese folosind ADDR

```
PROCEDURE PROCED (PARAM:INTEGER);
VAR REC: RECORD
      J: INTEGER;
      BOOL: BOOLEAN;
         END;
    R: REAL; S1;ARRAY [1..10] OF CHAR;
BEGIN
 WRITELN ('ADDR(PROCED)=',ADDR(PROCED));
 WRITELN ('ADDR(PARAM)=',ADDR(PARAM));
 WRITELN ('ADDR(REC)=',ADDR(REC));
 WRITELN ('ADDR(REC.J)=',ADDR(REC.J));
 WRITELN ('ADDR(R)=',ADDR(R));
 WRITELN ('ADDR(S1)=',ADDR(S1));
END;
```

6) Exemplu de utilizare @CMD

```
PROGRAM CMDLINE;
VAR S: STRING; PTR: ↑STRING; F: FILE OF CHAR;
EXTERNAL FUNCTION @CMD:↑STRING;
BEGIN
 PTR:=@CMD; S:=PTR↑;
 ASSIGN (F,S); RESET(F);
END.
```

7) Program de afişare la imprimantă a unor date de diferite tipuri (fişier de tip TEXT).

```
PROGRAM PRINTER;
VAR
 F:TEXT;  (* fisier imprimanta *)
 I:INTEGER; R:REAL;
 S:STRING;
BEGIN
 ASSIGN (F,'LST:');
 REWRITE (F);
 IF IORESULT=255 THEN
  WRITELN ('Eroare la imprimanta')
 ELSE
  BEGIN
   S:='Acesta este un sir';
   I:=55; R:=3.141563;
   WRITE (F,S)
   WRITELN (F);
   WRITELN (F,I:4,R:7:3);
   WRITELN (F,'Sfirsit test imprimanta')
  END
 END.
```

8) Program de creare şi de afişare secvenţială a unui fişier de caractere folosind procedurile standard GET şi PUT.

```
PROGRAM PUTGET,
 TYPE CHFILE=FILE OF CHAR;
 VAR OUTFILE   CHFILE; RESULT   INTEGER; FILENAME   STRING [16];

PROCEDURE WRITEFILE (VAR F : CHFILE);
 VAR CH : CHAR;
 BEGIN FOR CH   = '0' TO '9' DO
  BEGIN F^ := CH , PUT (F) END;
 END; {WRITEFILE}

PROCEDURE READFILE (VAR F : CHFILE),
 VAR I : INTEGER; CH : CHAR;
 BEGIN FOR I := 0 TO 9 DO
   BEGIN CH := F^ , GET (F) , WRITELN (CH) END;
 END; {READFILE}

BEGIN FILENAME := 'TEST.DAT',
 ASSIGN (OUTFILE,FILENAME); REWRITE (OUTFILE),
 IF IORESULT = 255 THEN WRITELN ('Eroare la creare',FILENAME)
  ELSE BEGIN WRITEFILE (OUTFILE); CLOSE (OUTFILE,RESULT);
    IF RESULT = 255 THEN WRITELN ('Eroare la inchidere',FILENAME)
     ELSE BEGIN WRITELN ('Inchidere corecta',FILENAME),
         RESET (OUTFILE);
         IF IORESULT = 255 THEN WRITELN ('Eroare la deschidere')
          ELSE READFILE (OUTFILE)
           END;
       END;
END
```

9) Program pentru scriere (citire) în acces direct, folosind procedurile SEEKWRITE şi SEEKREAD

```
program RANDFILE,
type PERSON=record NAME string[30],  ADDRESS string[30] end;
var RF:file of PERSON, S:string[30],
    I integer; ERROR:boolean, CH:char.

procedure ERRCHK,
begin ERROR:=TRUE,
 case IORESULT of
  0:begin writeln('Corect');ERROR:=FALSE end,
  1,3,5   writeln('Citire in afara fisierului'),
  2,4     writeln('Eroare sistem'),
 end;
end; {ERRCHK}

procedure READREC,
begin
 write('Numar inregistrare:'); readln(I);
 SEEKREAD(RF,I); ERRCHK;
 if ERROR then EXIT;
 writeln(RF^.NAME,'/',RF^.ADDRESS);
end; {READREC}
```

```
procedure WRITEREC,
begin
 write('Nume:');readln(S);
 RF^.NAME:=S;
 write('Adresa:'),readln(S),
 RF^.ADDRESS:=S;
 write('Nr.inregistrare:'), readln(I);
 SEEKWRITE(RF,I);ERRCHK;
end; {WRITEREC}

begin {MAIN}
 write('Nume fisier:'), readln(S),
 ASSIGN(RF,S), reset(RF);
 if IORESULT=255 then
  begin  rewrite(RF), CLOSE(RF,I), reset(RF) end;
 repeat
  write('Read/Write/Quit ,'); read(CH);writeln;
  case CH of
   'R':READREC,
   'W':WRITEREC,
   'Q':begin CLOSE(RF,I);EXIT end;
  else writeln ('Numai R/W/Q !');
  end;
 until CH='Q'
end.
```

10) Program de copiere fişier cu proceduri GET şi PUT

```
PROGRAM GETPUT;
TYPE
 CHFILE=FILE OF CHAR;
VAR
 A,B:CHFILE;
 NAME:STRING
PROCEDURE TRANSFER (VAR SRC:CHFILE; VAR DEST:CHFILE);
VAR
 RESULT:INTEGER;
BEGIN
 WHILE NOT EOF(SRC) DO
  BEGIN
   DEST↑ := SRC↑;
   PUT (DEST);
   GET (SRC);
  END;
 CLOSE (DEST,RESULT);
 IF RESULT=255 THEN
  WRITELN ('Eroare la inchidere fisier creat')
END;

 { Program principal pentru exemplele 10,11,12 }
```

```
BEGIN
 WRITE ('Sursa?'); READLN(NAME);
 ASSIGN (A,NAME);
 RESET (A);
 IF IORESULT=255 THEN
  BEGIN
   WRITELN ('Fisier inexistent:',NAME);
   EXIT
  END;
 WRITE ('Destinatie?');  READLN(NAME);
 ASSIGN (B,NAME);
 REWRITE(B);
 IF IORESULT=255 THEN
  BEGIN
   WRITELN('Nu mai este loc pentru fisierul',NAME);
   EXIT
  END;
 TRANSFER (A,B)
END.
```

11) Program de copiere a unui fişier bloc cu bloc

```
PROGRAM BYTECOPY;
 (* Foloseste proceduriIe GNB si WNB *)
CONST BUFSZ=2047;
TYPE
 PAOC=ARRAY[1..BUFSZ] OF CHAR;
 TFILE=FILE OF PAOC;
 CHFILE=FILE OF CHAR;
VAR
 A:TFILE;
 B:CHFILE;
 NAME:STRING;
PROCEDURE TRANSFER (VAR SRC:TFILE; VAR DEST:CHFILE);
VAR
 CH:CHAR;
 RESULT:INTEGER;
 ABORT:BOOLEAN;
BEGIN
 ABORT:=FALSE;
 WHILE (NOT EOF(SRC)) AND (NOT ABORT) DO
  BEGIN
   CH:=GNB(SRC);
   IF WNB (DEST,CH) THEN
     BEGIN
      WRITELN ('Eroare la scriere octet');
      ABORT:=TRUE;
     END;
   END;
 CLOSE (DEST,RESULT);
 IF RESLT=255 THEN
   WRITELN ('Eroare la inchidere')
END;
 (* Programul principal in exemplul 10 *)
```

12) Program de copiere a unui fișier octet cu octet

```
PROGRAM BYTECOPY;
 (* Foloseste procedurile GNB si WNB *)
CONST BUFSZ=2047;
TYPE
 PAOC=ARRAY[1..BUFSZ] OF CHAR;
 TFILE=FILE OF PAOC;
 CHFILE=FILE OF CHAR;
VAR
 A:TFILE;
 B:CHFILE;
 NAME:STRING;
PROCEDURE TRANSFER (VAR SRC:TFILE; VAR DEST:CHFILE);
VAR
 CH:CHAR;
 RESULT:INTEGER;
 ABORT:BOOLEAN;
BEGIN
 ABORT:=FALSE;
 WHILE (NOT EOF(SRC)) AND (NOT ABORT) DO
  BEGIN
   CH:=GNB(SRC);
   IF WNB (DEST,CH) THEN
     BEGIN
      WRITELN ('Eroare la scriere octet');
      ABORT:=TRUE;
     END;
   END;
 CLOSE (DEST,RESULT);
 IF RESLT=255 THEN
   WRITELN ('Eroare la inchidere')
END;
 (* Programul principal in exemplul 10 *)
```

13) Exemplu de program segmentat

```
{Fisierul PROG.PAS}
PROGRAM MAINPROG;
VAR I:INTEGER; CH:CHAR,
EXTERNAL [1] PROCEDURE OVL1;
EXTERNAL [2] PROCEDURE OVL2;
BEGIN
 REPEAT
  WRITE('Introduceti o litera: A/B/Q');
  READ(CH); CASE CH OF
   'A','a' : BEGIN I:=1; OVL1 END;
   'B','b' : BEGIN I:=2; OVL2 END
   ELSE; END {CASE}
 UNTIL CH IN ['Q','q'];
 WRITELN('Stop')
END.
```

```
    {Fisierul MOD1.PAS}
MODULE OVERLAY1;
VAR I:EXTERNAL INTEGER;
PROCEDURE OVL1;
BEGIN WRITELN('In segmentul 1 I=',I) END;
MODEND.

    {Fisierul MOD2.PAS}
MODULE OVERLAY2;
VAR I:EXTERNAL INTEGER;
PROCEDURE OVL2;
BEGIN WRITELN('In segmentul 2 I=',I) END;
MODEND.
```

Comenzile de linkeditare pentru acest program sînt:

LINKMT PROG, PASLIB/S/D:8000/V1: 4000/X:40
LINKMT PROG = PROG/O: 1, MOD1, PASLIB/S/P: 4000/L
LINKMT PROG = PROG/O:2, MOD2, PASLIB/S/P:4000/L

Observații

— valoarea 4000H pentru adresa zonei de suprapunere 1 este arbitrară și nu are legătura cu lungimea rădăcinii;

— valoarea 8000H pentru secțiunea de date a rădăcinii este tot arbitrară;

— opțiunea /L nu este necesară, dar permite punerea în evidență a faptului că la segmentele nerezidente nu se adaugă acele module din PASLIB care au fost deja introduse în rădăcină;

— fișierele create se numesc PROG.COM, PROG.001, PROG.002;

— executînd acest program prin comanda PROG se poate observa că, introducînd succesiv aceeași literă nu se reîncarcă segmentul respectiv de fiecare dată, dar introducînd secvența A, B, A, B,... se reîncarcă alternativ cele două segmente.

14) Exemplu de programe înlănțuite

```
    {Primul program - fisier CHAIN1.COM}
PROGRAM CHAIN1;
TYPE COMM=RECORD I,J:INTEGER END;
VAR GLOBALS:ABSOLUTE [$8000] COMM; CHFILE : FILE;
BEGIN
 WITH GLOBALS DO
  BEGIN I:=1; J:=-1 END;
 ASSIGN (CHFILE,'A:CHAIN2.COM'); RESET (CHFILE);
 IF IORESULT=255 THEN BEGIN
   WRITELN('Nu exista fisierul CHAIN2.COM'); EXIT END;
 CHAIN (CHFILE)
END.
```

```
     {Al doilea program - fisier CHAIN2.COM}
PROGRAM CHAIN2;
TYPE COMM=RECORD I,J:INTEGER END;
VAR GLOBALS: ABSOLUTE [$8000] COMM;
BEGIN
 WITH GLOBALS DO
  WRITELN ('I=',I,'J=',J);
END.
```

15) Exemple de proceduri și funcții în limbaj de asamblare: PEEK: citește din memorie un octet; POKE : scrie un octet în memorie

```
{fisier de tip .PAS}
PROGRAM PEEKPOKE;
TYPE BYTEPTR=^BYTE;
VAR ADDRESS:INTEGER; OPER:INTEGER;
    BBB:BYTE; PPP:BYTEPTR;
EXTERNAL PROCEDURE POKE (B:BYTE;P:BYTEPTR);
EXTERNAL FUNCTION PEEK (P:BYTEPTR):BYTE;
BEGIN REPEAT
 WRITE  ('Introduceti o adresa');
 READLN(ADDRESS); PPP:=ADDRESS;
 WRITE('Operatie:1=PEEK ,2=POKE');
 READLN (OPER);
 IF OPER=1 THEN WRITELN (ADDRESS,'contine',PEEK(PPP))
  ELSE IF OPER=2 THEN BEGIN
   WRITE('Introduceti un octet de date:');
   READLN (BBB); POKE (BBB,PPP) END
 UNTIL FALSE
END.

  ;fisier de tip .MAC
     PUBLIC     PEEK,POKE
PEEK:POP  B     ;scoate adresa de revenire in BC
     POP  H     ;valoare parametru in HL (BYTEPTR)
     MOV  E,M   ;continutul adresei de memorie in DE
     MVI  D,O
     PUSH D     ;rezultatul functiei in stiva
     PUSH B     ;adresa de revenire in stiva
     RET

POKE:POP  B     ;scoate adresa de revenire in BC
     POP  H     ;al doilea parametru (BYTEPTR) in HL
     POP  D     ;primul parametru in E (D=O)
     MOV  M,E   ;pune octet in memorie
     PUSH B     ;pune adresa de revenire in stiva
     RET
     END
```

10. PROGRAMARE ÎN LIMBAJUL FORTRAN SUB CP/M

Limbajul Fortran este unul dintre cele mai vechi limbaje de programare, dar și unul dintre cele mai actuale și mai folosite limbaje.

Perenitatea limbajului Fortran se explică în primul rînd prin simplitatea sa și prin adecvarea sa pentru exprimarea algoritmilor numerici de calcul.

Aplicațiile mai recente de grafică cu calculatorul și de proiectare asistată de calculator au readus și mai mult în atenția utilizatorilor acest limbaj.

Prin extinderile aduse limbajului, Fortran-ul poate fi și este utilizat cu succes și în aplicații nenumerice, de prelucrare a datelor, inclusiv cu fișiere disc.

Sistemul CP/M beneficiază de un compilator deosebit de performant, ceea ce adaugă un nou argument pentru folosirea acestui atît de cunoscut și de apreciat limbaj.

Compilatorul Fortran Microsoft implementează aproape complet standardul ANSI din 1966 (numit și Fortran IV), cu cîteva extinderi utile și cu unele restricții neimportante.

Timpul de compilare este foarte bun, datorită dimensiunii reduse a compilatorului, obținută pe seama unei analize mai sumare a sintaxei și semanticii programelor; de aceea mai pot apărea erori la execuție datorită unor erori de utilizare a limbajului, nesemnalate de compilator.

Atît codul obiect generat de compilator, cît și rutinele din biblioteca standard Fortran sînt foarte compacte, ceea ce conduce la programe executabile eficient ca memorie utilizată și ca timp.

Posibilitatea de a utiliza subprograme scrise în limbaj de asamblare poate deveni importantă, de exemplu în aplicațiile grafice, unde este necesar accesul la echipament, alături de calcule numeroase cu numere neîntregi.

Absenţa facilităţii de segmentare la linkeditare (cu L80) este resimţită în cazul programelor Fortran mai mari, în contextul memoriei limitate a microcalculatoarelor pe 8 biţi.

10.1. Utilizarea compilatorului F80

10.1.1. Opţiuni de compilare Fortran

Sînt posibile două moduri de utilizare a compilatorului F80, după cum se compilează unul sau mai multe fişiere de tip „.FOR":

— pentru o singură compilare:

```
F80 frel, fprn = ffor/opţ1/opţ2/...
```

— pentru mai multe compilări succesive:

```
F80
*frel, fprn = ffor/opţ1/opţ2...
*frel, fprn = ffor/opţ1/opţ2...
*^C
```

Ieşirea din ciclul de compilări succesive se face cu CTRL/C.

Fişierele de intrare şi de ieşire la compilare sînt:

frel fişier obiect relocabil, rezultat din compilare (are tipul implicit „.REL");

fprn fişier de listare a compilării (pe disc are tipul implicit „.PRN"). Ca fişiere de listare pot fi utilizate şi numele de dispozitive „TTY:" şi „LST:";

ffor fişier sursă cu programul FORTRAN (tipul implicit „.FOR")

Numele fişierelor frel, fprn, ffor au forma generală: d: nume.tip, cu menţiunea că discul suport „d" şi (sau) tipul pot lipsi. Compilatorul consideră în mod implicit că fişierele de ieşire frel şi fprn se creează pe acelaşi disc unde se află şi fişierul de intrare ffor.

Opţiunile de compilare FORTRAN (opţ1, opţ2,...) sînt:

/L se cere crearea unui fişier de listare (dacă acest fişier nu apare explicit în linia de comandă);

/H se cere afişarea în hexazecimal a adreselor de memorie (opţiune implicită);

/M se generează cod pentru a fi executat dintr-o memorie ROM.

/N se suprimă listarea codului generat de compilator în formă simbolică (în mod normal lista de compilare conţine codul maşină generat de compilator); se suprimă şi listarea informaţiilor despre alocarea memoriei pentru date şi instrucţiuni.

/O se cere afişarea în octal a adreselor de memorie;

/P se cere alocarea a încă 100 de octeţi pentru stiva utilizată de compilator (dacă se constată o depăşire a stivei la compilare);

/R se cere generarea unui fişier obiect relocabil (dacă acest fişier nu apare în mod explicit în linia de comandă).

Exemple de comenzi de compilare

1) F80 TESTF,LST: = TESTF/N

Se compilează programul din fişierul TESTF.FOR, se generează fişierul obiect TESTF.REL şi se afişează la imprimantă lista de compilare.

2) F80 ,TTY: = B:TESTF/N

Se face o verificare sintactică a programului din fişierul TESTF.FOR, cu afişare la consolă a întregului program, fără a se genera cod obiect.

3) F80 = B:TESTF/L/R/N

Această comandă este echivalentă cu comanda:

F80 B:TESTF,B:TESTF = B: TESTF/N

şi cu comanda:

F80 B:TESTF.REL,TESTF.PRN = B:TESTF.FOR/N

4) F80

```
* = FORT1/R
* = FORT2/R
*^C
```

La editarea de legături a modulelor compilate din Fortran trebuie să fie prezentă pe disc şi biblioteca standard FORLIB.REL, deoarece codul generat de compilator conţine multe apeluri de subrutine din această bibliotecă. Linkeditorul L80 conţine un nume de bibliotecă în care se caută automat pentru satisfacerea referinţelor externe; dacă acest nume este FORLIB, atunci nu este necesară folosirea lui în comandă, dar dacă acest nume este BASLIB (sau alt nume de bibliotecă) atunci numele FORLIB trebuie folosit în comandă cu opţiunea de căutare selectivă /S. Exemplu:

L80 PROG, SUB1, SUB2, FORLIB/S, PROG/N/E

Dacă se folosesc subrutine din alte biblioteci, atunci trebuie inclus numele acestor biblioteci în comanda de linkeditare, dar înaintea numelui bibliotecii standard FORLIB (care trebuie să fie ultimul fişier de intrare, dacă se foloseşte).

10.1.2. Erori la compilare şi la execuţie

Erorile sintactice semnalate de compilatorul F80 sînt de două tipuri:

— avertismente, marcate prin caracterul „%";
— erori grave, marcate prin caracterul „?".

În general, un program care are numai avertismente poate fi linkeditat şi executat.

Mesajele de eroare conţin unul dintre caracterele speciale menţionate, numărul liniei eronate, un text care explică cauza erorii şi ultimele 20 de caractere din linie dinaintea erorii.

Mesaje de eroare la execuția programelor FORTRAN

Avertismente

IB (Input Buffer limit exceeded)
depășire capacitate buffer de citire

TL (Too many Left parantheses in format)
prea multe paranteze în format

OB (Output Buffer limit exceeded)
depășire capacitate buffer de scriere

DE (Decimal Exponent overflow)
exponent mai mare ca 99

IS (Integer Size too large)
întreg prea mare

BE (Binary Exponent overflow)
exponent mai mare ca 38

IN (INput record too long)
înregistrare prea mare la citire

OV (arithmetic OVerflow)
depășire aritmetică

CN (CoNversion overflow)
depășire la conversia real-întreg

SN (argument to SiN too large)
argument prea mare la funcția SIN

A2 (both argument of ATAN2 are 0)
ambele argumente ale funcției ATAN2 sînt 0

IO (illegal I/O operation)
operație de I/E incorectă

BI (Buffer size exceeded during binary I/0)
depășire buffer la citire/scriere binară

RC (negative Repeat Count in format)
factor de repetiție negativ în format

Erori fatale

ID (Illegal format Descriptor)
descriptor de format incorect

FO (Format field width is 0)
lungime de cîmp nulă în format

MP (Missing Period in format)
lipsă punct în format

FW (format Field Width is too small)
lungime de cîmp prea mică în format

IT (I/O Transmission error)
eroare la transfer de I/E

ML (Missing Left parantheses in format)
absență paranteză stînga din format

DZ (Division by Zero, real or integer)
împărțire prin 0 la întregi sau reali

LG (illegal argument to LOG function)
argument ilegal la funcția LOG

SQ (illegal argument to SQRT function)
argument ilegal la funcția SQRT

DT (Data Type does not agree with format specification)
neconcordanță între tipul variabilelor și format

EF (EOF encountered on read)
sfîrșit de fișier la citire

10.2. Particularități de implementare Fortran sub CP/M

10.2.1. Diferențe Fortran-80 față de standard

Limbajul acceptat de F80 are următoarele *restricții* față de standard:
— nu există tipul de date complex;
— există tipul BYTE pentru întregi pe un octet;
— tipurile de date diferite se reprezintă pe lungimi diferite:
  — tipul logic pe un octet (8 biți));
  — tipul întreg pe 2 octeți (16 biți);
  — tipul real pe 4 octeți (32 biți);
  — tipul dublă precizie pe 8 octeți (64 biți).

Octetul inferior dintr-un întreg poate fi folosit ca variabilă de tip BYTE sau de tip logic în expresii sau la apelarea de subprograme.

— În caz de continuare a unei instrucțiuni DO pe două linii trebuie ca semnul de atribuire („=") și prima virgulă din instrucțiune să apară pe prima linie.

— În lista de intrare/ieșire a unei instrucțiuni READ sau WRITE nu sînt permise subliste de variabile în paranteze, dar ciclurile DO implicite în paranteze nu sînt considerate subliste.

— Lista descriptorilor din instrucțiunea FORMAT se reia de la început și nu de la ultima paranteză deschisă, atunci cînd sînt mai multe variabile în lista de intrare/ieșire decît descriptori de format. Reluarea se poate face de mai multe ori.

— Declarațiile neexecutabile trebuie să fie scrise la începutul unui program în ordinea următoare:

PROGRAM, SUBROUTINE, FUNCTION, BLOCK DATA
IMPLICIT
DIMENSION, INTEGER, REAL, LOGICAL, DOUBLE PRECISION, EXTERNAL
COMMON
EQUIVALENCE
DATA

Funcții - instrucțiune

— Un program principal FORTRAN poate fi precedat, opțional, de instrucțiunea:

PROGRAM nume

Dacă instrucțiunea lipsește, programul primește automat numele „$MAIN".

— Funcția BACKSPACE nu este recunoscută de compilatorul F80.

Limbajul Fortran-80 are *în plus față de standard* următoarele facilități:

— O instrucțiune se poate continua pe orice număr de linii.

— Se pot folosi constante hexazecimale de forma X'HHHH' sau Z'HHHH' oriunde sînt permise constante întregi (dacă sînt mai puțin de 4 cifre hexazecimale, atunci valoarea constantei se aliniază la dreapta pe 16 biți și se completează în stînga cu zerouri).

— În expresii se pot utiliza constante alfanumerice (unul sau două caractere între ghilimele) oriunde sînt permise constante întregi (caracterele sînt interpretate ca număr întreg).

— În expresii se pot utiliza variabile logice ca întregi pe un octet, cu valori între —127 și 128.

— Ca argument efectiv la apelarea de proceduri sau funcții se pot utiliza literali alfanumerici de orice lungime (șiruri de caractere încadrate de apostrof).

— Sînt permise expresii mixte și atribuiri mixte, iar conversiile de tip se realizează automat.

— Într-o instrucțiune DO se pot folosi ca variabile-contor și variabile întregi pe un octet (sau logice) pentru generarea unui cod mașină mai eficient.

— Într-o instrucțiune FORMAT, în instrucțiunea DATA sau în expresii un text constant se poate scrie între ghilimele simple sau precedat de „nH"; deci forma 'text' este echivalentă cu nHtext.

— Instrucțiunile READ și WRITE pot utiliza opțiunile „ERR=n" și „END = n", pentru salt automat la o secvență de tratare a erorilor și, respectiv, la o secvență de tratare a sfîrșitului de fișier.

— Funcțiile tip instrucțiune pot utiliza variabile indexate ca argumente formale.

— În instrucțiunile STOP c și PAUSE c, constanta „c" poate fi formată din 1—6 caractere ASCII (între ghilimele).

— Există următoarele subprograme standard suplimentare:

PEEK (a)

Funcție cu rezultat de un octet egal cu conținutul locației de memorie cu adresa „a";

POKE (a1, a2)

Subrutina care depune la adresa „a1" valoarea „a2";

INP (p)

Funcţie cu rezultat de un octet egal cu conţinutul portului de I/E cu numărul „p".

OUT (p, a)

Subrutină care trimite la portul „p" de I/E valoarea „a" de un octet.

10.2.2. Precizări asupra limbajului Fortran F80

— Compilatorul nu face distincţie între litere mari şi litere mici, dar se recomandă utilizarea de litere mari.

— Numele de variabile pot avea maxim 6 caractere alfabetice şi numerice.

— Caracterul „$" este tratat ca literă, dar se recomandă ca numele simbolice alese de programator să nu înceapă cu „$", pentru a nu coincide cu nume de variabile sau de subprograme din biblioteca FORLIB.

— Indicii variabilelor indexate nu pot fi decît expresii întregi pozitive cu una din formele următoare: c, v, v + c, v — c, c*v, c*v + c, c*v—c („c" este o constantă întreagă, iar „v" este o variabilă întreagă).

—Variabilele indice nu pot fi variabile indexate.

— Reprezentarea datelor:

— constantele alfanumerice au bitul cel mai semnificativ (bitul 7) din ultimul caracter pus pe 1; excepţie fac constantele alfanumerice din instrucţiunea DATA;

— constantele logice au următoarea reprezentare hexazecimală:

.FALSE. = 00, .TRUE. = FF sau orice valoare nenulă;

— variabilele logice se pot utiliza şi ca variabile întregi de un octet în expresii aritmetice sau în comparaţii;

— variabilele întregi pot avea valori între —32768 şi +32767;

— variabilele reale au o precizie de 7 cifre zecimale şi un exponent cuprins între —38 şi +38;

— variabilele reale cu precizie dublă au 16 cifre zecimale exacte şi acelaşi exponent maxim ca variabilele reale simplă precizie;

— sînt permise următoarele instrucţiuni pentru declararea tipului datelor:

INTEGER, REAL, LOGICAL, DOUBLE PRECISION
BYTE, INTEGER*1, INTEGER*2, REAL*4, REAL*8

Următoarele declaraţii sînt echivalente:

LOGICAL, BYTE, INTEGER*1
INTEGER, INTEGER*2
REAL, REAL*4
DOUBLE PRECISION, REAL*8

— Este permisă modificarea convenţiei implicite de tip printr-o instrucţiune de forma:

IMPLICIT tip (a,a1—a2...)

unde a, a1, a2 sînt litere; Exemple:

IMPLICIT INTEGER (A, W—Z), REAL (B—F, M—Q)

— Instrucţiunile ENCODE şi DECODE se pot utiliza pentru conversia datelor în memorie:

ENCODE (a, f) listă
DECODE (a, f) listă

unde: „a" este numele unui tablou;
„f" este eticheta instrucţiunii FORMAT;
„lista" este o listă de variabile similară cu lista din instrucţiunile READ/WRITE.

Instrucţiunea DECODE este similară cu instrucţiunea READ, dar converteşte caracterele ASCII din tabloul „a" în conformitate cu formatul specificat. Instrucţiunea ENCODE este similară cu instrucţiunea WRITE, dar converteşte valorile variabilelor din listă în ASCII şi depune rezultatul în tabloul „a".

— La instrucţiunea de atribuire se aplică următoarele reguli:

— atribuirea de expresii logice la o variabilă întreagă se face cu extinderea semnului în octetul superior;

— atribuirea de expresii întregi sau logice la o variabilă reală sau în dublă precizie se face folosind numărul real echivalent rezultatului logic sau întreg;

— atribuirea de expresii reale sau dublă precizie la variabile întregi sau logice se face prin trunchiere;

— atribuirea de expresii dublă precizie la variabile reale se face prin rotunjire.

— La execuţie nu sînt semnalate depăşirile la operaţii aritmetice cu întregi şi nici nu se fac verificări asupra indicilor variabilelor indexate.

— Operatorii logici .AND.,.OR.,.XOR. realizează operaţii logice bit cu bit cu operanzi de un octet sau de un cuvînt; operatorul .NOT. realizează complementul faţă de 1 (inversarea logică).

— Numele unui bloc comun trebuie să fie diferit de numele subprogramelor. Blocul comun blanc nu are nume, deci:

COMMON X,Y

este echivalent cu:

COMMON //X,Y

— Este posibil ca în subprograme să fie mai puţine variabile decît în programul principal într-un bloc comun, dar nu mai multe (dimensiunea blocului comun este determinată de instrucţiunea COMMON din programul principal).

— Mărimea unui bloc comun se poate extinde printr-o instrucțiune EQUIVALENCE ca în exemplul următor:

```
REAL R(2,2)
COMMON X,Y,Z
EQUIVALENCE (Z,R(3))
```

stabilește următoarele echivalențe:

```
X = R(1,1)
Y = R(1,2)
Z = R(2,1)
  = R(2,2)
```

A se observa că în instrucțiunea EQUIVALENCE se poate scrie R(3) în loc de R(2,1)

— Nu se pot echivala cu EQUIVALENCE două elemente din același tablou sau două elemente din blocuri COMMON.

— În instrucțiunea FORMAT sînt permise maximum două nivele de paranteze, inclusiv paranteza obligatorie la orice FORMAT. Se reamintește că reluarea descriptorilor de format se face de la începutul listei, deci de la prima paranteză deschisă.

— Sînt permise instrucțiuni READ și WRITE fără format pentru transferul datelor fără nici o conversie sau editare:

```
READ (n [,ERR = n1] [,END = n2]) listă
WRITE (n [,ERR = n1] [,END = n2]) listă
```

Folosind instrucțiuni de citire-scriere fără format se pot scrie înregistrări oricît de lungi; fiecare nouă înregistrare începe la un sector nou, iar sectorul anterior este completat cu zerouri.

— instrucțiunile READ și WRITE cu format nu pot scrie o înregistrare mai mare de 128 de octeți, deoarece zona buffer de ieșire are lungimea de 128.

— O instrucțiune READ nu trebuie să aibă o listă cu mai multe date decît capacitatea unei înregistrări, dar este posibilă citirea parțială a unei înregistrări.

— Înregistrările scrise într-un fișier printr-un program Fortran sînt separate între ele printr-un singur caracter ⟨CR⟩ și nu printr-o pereche de caractere ⟨CR⟩ ⟨LF⟩ ca în fișierele create cu un editor de texte.

— În instrucțiunile READ/WRITE cu format este permis ca, în locul etichetei instrucțiunii FORMAT, să se utilizeze un nume de tablou de tip întreg sau logic care să conțină șirul de caractere ce descrie formatul. Exemplu:

```
      INTEGER FRMT (4), TAB(3)
      READ(3,300)FRMT
600   FORMAT(4A2)
      READ(3,FRMT)TAB
```

— Specificatorii admiși în instrucțiunea FORMAT sînt:

rFw.d, rGw.d, rEw.d, rDw.d, rIw, rLw, rAw,
nHa1a2..an, 'a1a2..an', nX, mP

— La scriere cu specificatorul Ew.d este necesar ca w>d+7, deoarece se afişează întotdeauna o cifră zero la partea întreagă a mantisei.

— La scriere cu specificatorul Lw, dacă w>1, atunci se scriu w—1 spaţii înaintea literei „F" sau „T", care reprezintă valoarea variabilei logice afişate.

— Efectul specificatorului mP este de a stabili un factor de scalare „m" (m>0) pentru numerele reale introduse sau afişate: la citire numerele fără exponent se împart prin 10**m, iar la scriere numerele se înmulţesc cu 10**m înainte de afişare.

— Numerele logice ale dispozitivelor periferice folosite în instrucţiunile READ şi WRITE pot fi:

1: consola;
2: imprimanta;
3: consola;
4: nealocat;
5: nealocat;
6,...,10: fişierele secvenţiale pe discul A cu numele

FORT06.DAT,...FORT10.DAT

Semnificaţia acestor numere logice poate fi modificată prin subrutina standard OPEN.

— Datele numerice citite de la consolă se pot introduce într-un format liber, separînd numerele prin virgule sau spaţii.

— Este posibil ca datele unui program Fortran să fie introduse în linia de comandă prin care se apelează programul.

— După scrierea unui mesaj la consolă cursorul rămîne poziţionat după ultimul caracter şi nu trece automat pe linia următoare.

— Se pot crea şi exploata atît fişiere secvenţiale, cît şi fişiere în acces direct.

— Dacă se scrie într-un fişier secvenţial existent, atunci se şterge conţinutul anterior al fişierului (nu este posibilă actualizarea unui fişier).

— Pentru fişierele secvenţiale prima instrucţiune READ sau WRITE deschide automat fişierul, iar închiderea fişierului se face la STOP sau printr-o instrucţiune ENDFILE n.

— Instrucţiunea REWIND n închide fişierul cu numărul logic „n" şi apoi îl redeschide.

— Subrutina OPEN permite deschiderea explicită a unui fişier pentru acces secvenţial sau direct, cu atribuirea unui nume şi unui număr logic fişierului (numărul logic trebuie să fie cuprins între 1 şi 10). Subrutina OPEN primeşte ca argumente:

— numărul logic asociat fişierului;

— numele și tipul fișierului pe 8 și respectiv pe 3 caractere (eventual completate cu blancuri);

— numărul unității de disc pe care se află fișierul:

0 pentru discul implicit

1 pentru discul A:

2 pentru discul B: ş.a.m.d.

Exemplu:

CALL OPEN (6,'PLOT   DAT',2)

Subrutina OPEN nu este necesară dacă se creează sau se exploatează un fișier secvențial cu număr logic între 6 și 10, sub numele implicit FORTxx.DAT, unde „xx" = 06, 07, 08, ..10.

Dacă numele fișierului folosit în OPEN se citește de la consolă, el trebuie introdus sub aceeași formă: 11 caractere, fără punct între nume și extensie (sau este adus prin program la această formă).

— Instrucțiunile READ și WRITE pentru acces direct au forma următoare:

READ (u,f,REC = n,ERR = m) listă

unde:

u numărul logic asociat fișierului (prin OPEN sau implicit);

f eticheta instrucțiunii FORMAT;

n numărul articolului citit/scris (un articol are 128 octeți deci coincide cu un sector disc);

m eticheta instrucțiunii la care se sare în caz de eroare la citire sau la scriere.

— Prin scriere în acces direct un fișier se poate extinde.

— Tablourile din subprograme pot fi declarate cu dimensiuni variabile, cu condiția ca atît numele tabloului, cît și numele variabilelor dimensiune să fie transmise ca argumente.

— Inițializarea variabilelor din blocuri comune se poate face numai în subprograme de tip BLOCK DATA.

10.2.3. Convenția de apelare a subprogramelor FORTRAN

Transmiterea de date între un program Fortran și un subprogram scris în limbaj de asamblare se poate face în două moduri:

— prin lista de argumente;

— prin blocuri comune (instrucțiunea COMMON).

Pentru legarea de subprograme scrise în limbaj de asamblare cu programe FORTRAN este necesară cunoașterea convențiilor de comunicare cu subprogramele apelate.

Transmiterea parametrilor la subprogramul apelat:

Indiferent de tipul parametrilor, se transmit adresele de memorie ale acestor parametri prin registre și, eventual, prin memorie după cum urmează:

a) Dacă numărul de argumente este mai mic sau egal cu 3, atunci adresele argumentelor se transmit prin registre după cum urmează:

adresa argumentului 1 în registrele HL;
adresa argumentului 2 în registrele DE;
adresa argumentului 3 în registrele BC;

b) Dacă numărul de argumente este mai mare ca trei, atunci adresele argumentelor se transmit astfel:

adresa argumentului 1 în registrele HL;
adresa argumentului 2 în registrele DE;
adresele argumentelor 3, 4,... printr-o zonă de memorie a cărei adresă se află în registrele BC (adresa din BC este adresa octetului inferior din argumentul 3).

Numărul de argumente nu se transmite explicit la subprogram, deci trebuie cunoscut de subprogram.

Rezultatul subprogramelor de tipul funcţiei se transmite prin registre, în funcţie de tipul rezultatului:

întregi : valoare rezultat în registrele HL;
reali : valoare rezultat în \$AC;
dublă precizie : valoare rezultat în \$DAC.

Variabilele globale \$AC şi \$DAC reprezintă zone de memorie utilizate ca acumulator pentru operaţii cu numere reale şi, respectiv, numere în precizie dublă.

10.3. Funcţii şi subrutine standard în FORTRAN-80

10.3.1. Funcţii intrinseci

Limbajul Fortran-80 recunoaşte următoarele funcţii de bibliotecă (cuprinse în fişierul FORLIB.REL):

Nume	Definiţie	Tip argument	Tip funcţie
ABS	valoare absolută	real	real
IABS		întreg	întreg
DABS		dublă precizie	dublă precizie
AINT	partea întreagă	real	real
INT		întreg	întreg
IDINT		dublă precizie	întreg
AMOD	a1 mod a2	real	real
MOD		întreg	întreg
DMOD		dublă precizie	dublă precizie
AMAX0	max (a1, a2,...)	întreg	real
AMAX1		real	real
MAX0		întreg	întreg
MAX1		real	întreg

DMAX1		dublă precizie	dublă precizie
AMIN0	min (a1, a2,...)	întreg	real
AMIN1		real	real
MIN0		întreg	întreg
MIN1		real	întreg
DMIN1		dublă precizie	dublă precizie
FLOAT	întreg ⇒ real	întreg	real
IFIX	real ⇒ întreg	întreg	real
SIGN	sign(a)	real	real
ISIGN		întreg	întreg
DSIGN		dublă precizie	dublă precizie
DIM	a1-min(a1, a2)	real	real
IDIM		întreg	întreg
SNGL	dublă ⇒ simplă	dublă precizie	real
DBLE	simplă ⇒ dublă	real	dublă precizie
EXP	e**a	real	real
DEXP		dublă precizie	dublă precizie
ALOG	logaritm natural	real	real
DLOG		dublă precizie	dublă precizie
ALOG10	logaritm zecimal	real	real
DLOG10		dublă precizie	dublă precizie
SIN	sinus	real	real
DSIN		dublă precizie	dublă precizie
COS	cosinus	real	real
DCOS		dublă precizie	dublă precizie
TANH	tanh(a)	real	real
SORT	rădăcina pătrată	real	real
DSQRT		dublă precizie	dublă precizie
ATAN	arctan(a)	real	real
DATAN		dublă precizie	dublă precizie
ATAN2	arctan(a1/a2)	real	real
DATAN2		dublă precizie	dublă precizie

Cu a, a1, a2,... s-au notat argumentele funcției. Numărul de argumente al unei funcții reiese din definiție.

10.3.2. Utilizarea subprogramelor din biblioteca FORLIB.REL

Biblioteca FORLIB.REL conține subprograme utilizate în mod normal de programele rezultate din compilări FORTRAN. O serie de subprograme din bibliotecă sînt de utilitate mai generală și pot fi apelate de programe scrise în limbaj de asamblare; aceste subprograme pot fi clasificate în următoarele categorii:

a) subrutine pentru operaţii aritmetice;
b) subrutine pentru conversii de tip;
c) subrutine de citire-scriere cu format;
d) funcţii FORTRAN intrinseci (funcţii standard).

Convenţiile de transmitere a argumentelor la aceste subprograme sînt direrite de convenţia de transmitere la apelarea de subprograme FORTRAN.

Subrutine pentru operaţii aritmetice

a) argumentul 1 în registre:
   — întregi în HL;
   — reali în $AC;
   — dublă precizie în $DAC.
b) argumentul 2 în registre sau în memorie, în funcţie de tip:
   — întregi în HL sau DE dacă HL conţin primul argument;
   — reali şi dublă precizie în memorie, la adresa dată de HL.

Variabilele globale $AC şi $DAC sînt definite în FORLIB:

$AC acumulator de virgulă mobilă în precizie simplă ($AC + 3 = adresă exponent);

$DAC acumulator de virgulă mobilă în precizie dublă ($DAC + 7 = adresă exponent).

Lista subrutinelor pentru operaţii aritmetice.

Funcţie	Nume	Arg.1	Arg.2
Adunare	$AA	R ($AC)	I (H,L)
	$AB	R ($AC)	R ((H,L))
	$AQ	D ($DAC)	I (H,L)
	$AR	D ($DAC)	R ((H,L))
	$AU	D ($DAC)	D ((H,L))
Scădere	$SA	R ($AC)	I (H,L)
	$SB	R ($AC)	R ((H,L))
	$SQ	D ($DAC)	I (H,L)
	$SR	D ($DAC)	R ((H,L))
	$SU	D ($DAC)	D ((H,L))
Înmulţire	$M9	I (H,L)	I (D,E)
	$MA	R ($AC)	I (H,L)
	$MB	R ($AC)	R ((H,L))
	$MQ	D ($DAC)	I (H,L)
	$MR	D ($DAC)	R ((H,L))
	$MU	D ($DAC)	D ((H,L))
Împărţire	$D9	I (H,L)	I (D,E)
	$DA	R ($AC)	I (H,L)
	$DB	R ($AC)	R ((H,L))
	$DQ	D ($DAC)	I (H,L)
	$DR	D ($DAC)	R ((H,L))
	$DU	D ($DAC)	D ((H,L))

Ridicare la putere	$ED	I (H,L)	I (D,E)
	$EA	R ($AC)	I (H,L)
	$EB	R ($AC)	R ((H,L))
	$EQ	D ($DAC)	I (H,L)
	$ER	D ($DAC)	R ((H,L))
	$EU	D ($DAC)	D ((H,L))

La operaţia de împărţire întreagă ($D9) registrele HL trebuie să conţină împărţitorul, iar registrele DE trebuie să conţină deîmpărţitul. După împărţire registrele HL conţin cîtul întreg, iar registrele DE conţin restul împărţirii întregi.

Subrutine pentru încărcarea şi memorarea variabilelor globale $AC, $DAC:

$L1 încărcare în $AC de la adresa din HL
$L3 încărcare în $DAC de la adresa din HL
$T1 memorare din $AC la adresa din HL
$T3 memorare din $DAC la adresa din HL

Convenţii la subrutine pentru *conversii de tip*

argument logic în A
argument întreg în HL
argument real în $AC
argument dublă precizie în $DAC

Lista subprogramelor de conversie:

Nume	Argument	Rezultat	Tip conversie
$CA	I (H,L)	R ($AC)	întreg-real
$CC	I (H,L)	D ($DAC)	întreg-dublă precizie
$CH	R ($AC)	I (H,L)	real-întreg
$CJ	R ($AC)	L (A)	real-logic
$CK	R ($AC)	D ($DAC)	real-dublă precizie
$CX	D ($DAC)	I (H,L)	dublă precizie-întreg
$CY	D ($DAC)	R ($AC)	dublă precizie-real
$CZ	D ($DAC)	L (A)	dublă precizie-logic

S-au folosit următoarele notaţii:

I = întreg;
R = real;
D = dublă precizie;
L = logic.

Subprograme pentru *citire şi scriere cu format*.

Pentru a programa în limbaj maşină operaţii de citire sau de scriere cu format trebuie utilizate următoarele tipuri de subrutine din FORLIB:

— Subrutine pentru iniţializarea unei citiri sau scrieri:

$W2 iniţializare WRITE cu 2 parametri

$W5 inițializare WRITE cu 5 parametri
$R2 inițializare READ cu 2 parametri
$R5 inițializare READ cu 5 parametri
— Subrutine pentru transferul valorilor în zona tampon de I/E:
$I0 transfer de întregi
$I1 transfer de reali
$I2 transfer de valori logice
$I3 transfer de valori în dublă precizie
— O subrutină de terminare a procesului de intrare-ieșire:
$ND

Parametrii subrutinelor $W2 și $R2 se transmit astfel:

HL adresa numărului logic al dispozitivului de I/E
DE adresa șirului format (ceea ce urmează cuvîntului FORMAT)

Ceilalți trei parametri ai subrutinelor $W5 și $R5 corespund parametrilor END, ERR și REC din instrucțiunile READ și WRITE; pentru transmiterea lor se folosesc registrele BC.

Parametrii subrutinelor $In se transmit astfel:

HL adresa numărului de dimensiuni al variabilelor din listă
DE adresa primei valori din lista care se transferă
BC adresa celei de a doua valori (dacă sînt numai două) sau adresa unei liste de adrese ale celorlalte valori.
A numărul de parametri.

Funcții FORTRAN intrinseci (funcții standard)

Parametrii din HL și DE (al treilea argument în BC) reprezintă adresele argumentelor.

Pentru MIN și MAX numărul de argumente se transmite în A.

Cu excepția funcțiilor INP, OUT, POKE, nici o funcție nu admite argumente de un octet.

10.4. Exemple de programe Fortran

1) Program pentru desenarea unei histograme cu bare orizontale (utilizare de constante de tip caracter)

```
      PROGRAM HISTO
C TABLOUL K CONTINE VALORILE HISTOGRAMEI
C TABLOUL LINIE CONTINE BLANCURI SI STELUTE
      INTEGER K(40)
      BYTE LINIE(80)
C LAT ESTE LATIMEA UNEI BARE DIN HISTOGRAMA
      LAT=2
C CITIRE DATE INITIALE
      READ (1,10) N,KMAX,(K(I),I=1,N)
10    FORMAT(16I5)
C DESENARE HISTOGRAMA LINIE CU LINIE
      DO 1 I=1,N
       NA=K(I)*80/KMAX+1
       DO 2 J=1,80
```

```
2           LINIE(J)=' '
           DO 3 J=1,NA
3           LINIE(J)='*'
           DO 4 L=1,LAT
4           WRITE(1,11) LINIE
1         CONTINUE
11        FORMAT(1X,80A1)
          END
```

2) Program de extragere a rădăcinii pătrate, cu calcule în dublă precizie

```
          PROGRAM DOUBLE
          IMPLICIT DOUBLE PRECISION (D,R)
          WRITE(3,200)
200       FORMAT(' INTRODUCETI NUMARUL
1         READ(1,10) DAT
10        FORMAT(D20 10)
C CALCUL CU FUNCTIE DE BIBLIOTECA
          RAD1=DSQRT(DAT)
C CALCUL CU FUNCTIE UTILIZATOR
          RAD2=DBLSQR(DAT)
          WRITE(3,11) DAT,RAD1,RAD2
11        FORMAT(3G25 15)
          GOTO1
          END
C
          DOUBLE PRECISION FUNCTION DBLSQR(X)
          DOUBLE PRECISION X,EPS,R1,R2
          EPS=1D-8
          R2=X
1         R1=R2
          R2=(R1+X/R1)/2
          IF(DABS((R2-R1)/R1) GT.EPS)GOTO1
          DBLSQR=R2
          RETURN
          END
```

3) Utilizare operații logice la nivel de bit cu întregi și constante hexazecimale

```
PROGRAM BIT
L1=X'AA'
L2=X'66'
L1NOT=.NOT. L1
LAND=L1.AND.L2
LOR=L1.OR.L2
LXOR=L1.XOR.L2
WRITE(3,1) L1,L2,L1NOT,LAND,LOR,LXOR
FORMAT(1X,2I8/1X,4I8)
END
```

4) Utilizare funcţii PEEK şi POKE

```
          PROGRAM PIKPOK
          BYTE B1,B2,B3
          B1=PEEK(0)
          B2=PEEK(X'C000')
          CALL POKE(X'C000',33)
          B3=PEEK(X'C000')
          WRITE(3,10) B1,B2,B3
10        FORMAT(3I6)
          END
```

5) Program de listare fişier text

```
          BYTE LINIE(72),NUME(11)
          READ(1,9)NUME
9         FORMAT(11A1)
          CALL OPEN(7,NUME,0)
          READ(7,1)LINIE
1         FORMAT(80A1)
          WRITE(1,2) LINIE
2         FORMAT(1X,80A1)
3         READ(7,1,END=9)LINIE
          WRITE(1,2)LINIE
          GOTO 3
9         STOP
          END
```

6) Program de salvare şi citire fără format a unei matrice într-un fişier secvenţial predefinit (FORTO6.DAT)

```
          INTEGER  A(10,10),B(10,10)
          DO 1 I=1,10
           DO 1 J=1,10
            A(I,J)=10*(I-1)+J
1         CONTINUE
          WRITE(6) A
          ENDFILE 6
          READ(6) B
          WRITE(3,3) ((B(I,J),J=1,10),I=1,10)
3         FORMAT(1X,10I8)
          END
```

7) Program de scriere-citire fişier în acces direct

```
        BYTE OP,REC(80),LR,LQ,LW
        DATA LR/'R'/,LQ/'Q'/,LW/'W'/
        CALL OPEN(6,'TEST    DAT',0)
1       WRITE(3,12)
12      FORMAT(' Operatie (Read/Write/Quit):')
        READ(1,11) OP
11      FORMAT(A1)
        IF (OP.EQ.LQ) STOP
        IF (OP.NE.LW .AND. OP.NE.LR) GOTO 1
        WRITE(3,13)
13      FORMAT(' Numar Inregistrare (1-2 cifre):')
        READ(1,14)N
14      FORMAT(I2)
        IF (OP.EQ.LR) GOTO 2
        WRITE (3,15)
15      FORMAT(' Introduceti date (max 80 car.):'/)
        READ(1,16) REC
16      FORMAT(80A1)
        WRITE(6,REC=N) REC
        GOTO 1
2       READ(6,REC=N,ERR=9) REC
        WRITE(3,17) REC
17      FORMAT(1X,80A1)
        GOTO 1
9       WRITE(3,18)
18      FORMAT(' Inregistrare in afara fisierului'/)
        GOTO 1
        END
```

8) Program Fortran cu subrutină în limbaj de asamblare; datele se transmit prin bloc comun neetichetat (comun blanc)

```
C COMUNICARE FORTRAN-ASAMBLOR PRIN ARGUMENTE
        INTEGER*1 I,J,K
10      READ(1,1) I,J
1       FORMAT(2I3)
        IF(I.EQ.0 .AND. J.EQ.0) STOP
        CALL BIC(I,J,K)
        WRITE(3,2)I,J,K
2       FORMAT(11X,I3,' and',I3,' =',I3/)
        GOTO 10
        END
```

```
; Comunicare Asamblor -Fortran prin bloc comun
      COMMON / /
I:    DS   1
J:    DS   1
K:    DS   1
      CSEG
      ENTRY  BIC
BIC:  LDA   I    ;primul operand
      MOV   B,A
      LDA   J    ;al doilea operand
      ANA   B    ;produs logic in A
      STA   K    ;pune rezultat in blocul comun
      RET
      END
```

9) Program Fortran cu subrutină în limbaj de asamblare; datele se transmit prin lista de parametri

```
C COMUNICARE FORTRAN-ASAMBLOR PRIN BLOC COMUN
         LOGICAL I,J,K
         COMMON  I,J,K
10       READ(1,1) I,J
1        FORMAT(2I3)
         IF(I+J .EQ. 0) STOP
         CALL  BIC
         WRITE(3,2) K
2        FORMAT(11X,'Produsul logic =',I3/)
         GOTO 10
         END
```

```
; Comunicare asamblor-Fortran prin argumente
      ENTRY  BIC
BIC:   MOV   A,M ;valoarea primului parametru
       XCHG      ;HL=valoare parametru 2
       ANA   M   ;produs logic cu parametrul 2
       STAX  B   ;rezultat in parametrul 3
       RET
       END
```

Exemplele anterioare sînt pur didactice, deoarece operatorul logic .AND. din F80 realizează produs logic bit cu bit între operanzi și deci nu este necesară o subrutină în limbaj mașină.

10) Utilizarea subrutinelor aritmetice din FORLIB pentru realizarea împărțirii D3 = D1/D2 în dublă precizie:

```
      EXTRN $L3, $D3, $DB
DIV:  LXI  H,D1
      CALL $L3        ;incarca D1 in $DAC
      LXI  H,D2
      CALL $DB        ;impartire $DAC la D2 cu rezultat in $DAC
      LXI  H,D3
      CALL $T3        ;memorare rezultat din $DAC in D3
      RET
      END
```

11) Utilizarea subrutinelor de scriere cu format din FORLIB:

```
     EXTRN      $W2, $IO, $ND
     ENTRY      TST
TST: LXI  H,LUN      ;numar logic dispozitiv de afisare
     LXI  D,FORMAT   ;adresa sir format
     CALL $W2
     LXI  H,DIMENS   ;numar de dimensiuni pentru variabila
     LXI  D,NUMAR    ;adresa numar afisat
     MVI  A,2        ;numar de parametrii $IO
     CALL $IO        ;transfer in zona de I/E
     CALL $ND        ;terminare I/E
     RET
FORMAT:   DB   '(16HRezultatul este:,I5)
LUN:      DW   1
DIMENS:   DW   1
NUMAR:    DW   9999
     END
```

11. PROGRAMARE ÎN LIMBAJUL BASIC SUB CP/M

Limbajul BASIC a apărut ca o variantă simplificată a limbajului FORTRAN, utilizabilă în mod interactiv și destinată învățării rapide a bazelor programării.

Ulterior limbajul BASIC a fost dezvoltat pentru lărgirea ariei sale de folosire, ajungînd astăzi un limbaj mai complex decît FORTRAN-ul, dar a rămas unul dintre cele mai atrăgătoare limbaje, în special prin modul de lucru conversațional.

Avantajul limbajului BASIC constă în ușurința cu care se poate experimenta în programare, modificînd continuu programele și obținînd rapid efectul acestor modificări, calitate deosebit de importantă în procesul de învățare a programării, în aplicații grafice și în crearea de diverse jocuri.

Timpul de răspuns redus rezultă pe de o parte din interpretarea instrucțiunilor BASIC (nu se mai generează cod mașină), iar pe de altă parte din integrarea unor facilități de editare și unor comenzi utilitare în limbaj. BASIC nu este doar un limbaj, el este primul „mediu de programare" (apărut chiar înaintea acestui termen), dar un mediu simplu care poate fi implementat și pe mașini foarte modeste ca posibilități.

Introducerea calculatoarelor personale a contribuit mult la răspîndirea limbajului BASIC, dar și la diversificarea sa (în special pe partea de comenzi grafice și acustice), astfel încît există aproape tot atîtea variante de BASIC cîte tipuri de calculatoare.

În prezent limbajul BASIC este de obicei primul limbaj de programare pentru mulți tineri, dar și un limbaj cu suficiente posibilități care să permită realizarea de aplicații diverse într-un timp foarte scurt comparativ cu alte limbaje compilate.

În sistemul CP/M—80 cel mai cunoscut și mai folosit BASIC este cel al firmei Microsoft, care a realizat un interpretor și un compilator compatibile pentru o versiune de limbaj foarte apropiată de standardul

existent. De asemenea, circulă cîteva versiuni „grafice" ale interpretorului BASIC Microsoft, extins cu cîteva comenzi grafice, dar aceste versiuni sînt specifice fiecărui tip de microcalculator.

11.1. Utilizarea interpretorului și compilatorului BASIC

11.1.1. Utilizarea interpretorului MBASIC

Numele original al fișierului cu interpretorul Microsoft este MBASIC.COM, dar de multe ori apare sub numele BASIC.COM.

Formele uzuale de apelare a interpretorului BASIC sînt:

BASIC

BASIC fișier

Cea de a doua formă încarcă interpretorul BASIC, care apoi încarcă fișierul cu numele ⟨fișier⟩ (de tip implicit „BAS") și lansează în execuție programul conținut în acest fișier; același efect se poate obține prin comanda „RUN fișier", după încărcarea interpretorului cu prima formă a comenzii.

Forma cea mai generală a comenzii de apelare a interpretorului BASIC este următoarea:

BASIC [fișier] [/F: maxfil] [M: maxmem] [/S: maxrec]

unde:

⟨maxfil⟩ este numărul maxim de fișiere de date simultan deschise (implicit sînt 3 fișiere); fiecare fișier suplimentar necesită 166 octeți de memorie pentru zonele de lucru;

⟨maxmem⟩ este limita superioară a memoriei pe care o poate folosi interpretorul BASIC (implicit este limita superioară a zonei TPA); această opțiune este necesară numai la folosirea de subrutine în limbaj de asamblare din BASIC;

⟨maxrec⟩ este lungimea maximă a înregistrărilor din fișierele de date (implicit 128).

Reîntoarcerea în sistemul CP/M se face cu comanda SYSTEM.

După ce a fost lansat, BASIC tipărește mesajul „Ok" care arată că interpretorul este gata să accepte comenzi.

Interpretorul MBASIC poate fi utilizat în două moduri: mod direct (imediat) și mod indirect (programat).

În mod direct instrucțiunile și comenzile nu sînt precedate de un număr de linie și sînt executate imediat ce sînt introduse.

Instrucțiunile nu sînt memorate, dar rezultatele operațiilor aritmetice și logice rămîn memorate în variabile.

Acest mod de lucru este util pentru depanare de programe și la folosirea interpretorului BASIC drept „calculator de buzunar" pentru operații rapide care nu necesită un program complet.

Modul indirect este folosit pentru introducerea de programe. Liniile de program trebuie precedate de cîte un număr, care determină și ordinea lor de execuție. Programul introdus rămîne în memorie pînă la ștergerea sa printr-o comandă. Este posibilă și salvarea pe disc, într-un fișier, a programului din memorie.

11.1.2. Utilizarea compilatorului BASCOM

Fișierele necesare pentru compilare BASIC sînt:

BASCOM.COM compilator BASIC
BASLIB.REL biblioteca de module relocabile pentru BASIC
BRUN.COM fișier necesar la execuția programelor BASIC
L80.COM linkeditor, versiunea 3.44.

Utilizarea compilatorului BASCOM este asemănătoare cu utilizarea compilatorului Fortran F80.

Forme de apelare:

1) BASCOM frel, fprn = fbas/opt
2) BASCOM
   *frel, fprn = fbas/opt

Ambele forme sînt echivalente și realizează o compilare.
„frel" este fișierul obiect de tipul „.REL", creat de compilator.
„fprn" este fișierul listing de tipul „.PRN", creat de compilator.
„fbas" este fișierul sursă de tipul „.BAS", citit de compilator.

Ca fișiere de listare se pot utiliza și dispozitivele TTY:, LST:

Opțiuni de compilare „opt":

/N nu se listează codul mașină generat de compilator
/R generează fișier de tipul „.REL"
/L generează fișier de tipul „.PRN"

Exemple:

1) BASCOM , TTY: = TEST/N
2) BASCOM TEST = TEST
3) BASCOM TEST, TEST = TEST/N
4) BASCOM = TEST/R/L
5) BASCOM
   * = TEST

Fișierele relocabile create de BASCOM trebuie legate împreună cu biblioteca BASLIB folosind versiunea 3.44 a linkeditorului L80. Exemple:

L80 TEST, BASLIB/S/G
L80 TEST, BASLIB/S=TEST/N/E

Pentru executarea unui program provenit din compilare BASIC este necesar să existe și fișierul BRUN.COM pe același disc cu fișierul de tipul „.COM" creat de L80.

Fişierele sursă pentru compilatorul BASCOM au acelaşi format ca şi pentru interpretorul BASIC, dar fiecare linie sursă trebuie precedată de un număr de linie. Aceste fişiere pot fi create fie cu un editor de texte, fie cu interpretorul şi salvate în format ASCII (SAVE „nume", A).

Limbajul BASIC acceptat de compilatorul BASCOM este în general acelaşi cu limbajul acceptat de interpretor, cu următoarele diferenţe:

— Nu sînt permise comenzile specifice modului interactiv: AUTO, CONT, DELETE, EDIT, LIST, LLIST, LOAD, MERGE, NEW, RENUM, SAVE.

Există o serie de funcţii predefinite noi, în special pentru numere reale în dublă precizie.

11.1.3. Comenzi ale interpretorului MBASIC

Din punct de vedere sintactic aceste comenzi fac parte din limbaj, ca şi instrucţiunile de prelucrare, de control şi de intrare-ieşire, fiind şi ele supuse standardizării. Din punct de vedere funcţional aceste comenzi, împreună cu unele caractere de control, sînt folosite de utilizator pentru a gestiona şi controla programele BASIC propriu-zise; ele pot fi grupate în comenzi utilitare şi comenzi de editare a programelor sursă.

Astfel sînt prevăzute comenzi pentru execuţia şi întreruperea unui program, pentru listarea la consolă sau la imprimantă a unui program, pentru ştergerea de linii, pentru modificarea selectivă de linii, pentru numerotarea şi renumerotarea automată a liniilor de program, pentru salvarea pe disc şi citirea de pe disc în memorie a unor programe etc.

Rolul special al comenzilor BASIC faţă de instrucţiunile BASIC rezultă şi din faptul că ele au sens şi sînt recunoscute numai de interpretor, în timp ce compilatorul acceptă numai instrucţiuni; de asemenea, comenzile nu trebuie precedate de numere de linie.

În procesul de introducere şi de editare a programelor sursă se folosesc următoarele caractere de control:

CTRL/A se trece în mod de lucru EDIT pe ultima linie introdusă;
CTRL/H şterge ultimul caracter tastat;
CTRL/I tabulare fixată din 8 în 8 coloane;
CTRL/R reafişează linia tastată;
CTRL/U şterge linia curentă.

În execuţia programelor BASIC se pot folosi următoarele caractere de control:

CTRL/C se întrerupe execuţia unui program şi se dă controlul interpretorului;
CTRL/G, generează un semnal sonor la terminal;
CTRL/O suspendă o aşteptare de I/E şi continuă execuţia programului; următorul CTRL/O activează I/E;

CTRL/S suspendă execuția programului;
CTRL/Q reia execuția după o suspendare cu CTRL/S;

Atribuirea de numere liniilor de program se poate face de către programator sau, în mod automat de către interpretor, dacă se folosește comanda AUTO. Este posibilă și renumerotarea automată a unor linii de program prin comanda RENUM.

Afișarea pe ecran a unui program sau a unor linii de program se poate face prin comanda LIST, iar listarea la imprimantă prin comanda LLIST.

Inserarea de noi linii în programul existent se face prin alegerea corespunzătoare a numărului liniei introduse, deoarece ordinea de execuție (și de listare) este determinată numai de ordinea numerelor de linie, indiferent de ordinea în care au fost introduse liniile.

Ștergerea de linii se poate face prin comanda DELETE sau prin reintroducerea numărului liniei urmat de <CR>.

Modificarea conținutului unor linii scurte se face simplu, reintroducînd instrucțiunea corectă cu același număr ca linia greșită, de cîte ori este nevoie.

În cazul liniilor lungi poate fi însă preferabilă utilizarea comenzii EDIT, care permite modificarea selectivă a caracterelor dintr-o linie.

De reținut că înainte de a introduce un nou program de la consolă trebuie folosită comanda NEW pentru ștergerea programului existent în memorie.

Execuția programului din memorie în întregime sau execuția unor secvențe din program se poate face folosind comanda RUN.

Întreruperea de către operator a execuției unui program și revenirea la nivel de comenzi interpretor se face prin caracterul CTRL/C. Continuarea unui program întrerupt se face prin comanda CONT. Întreruperea execuției unui program poate avea loc și ca urmare a unei erori de program adevărate sau simulate (prin comanda ERROR), iar reluarea execuției după tratarea unei erori se face prin comanda RESUME.

Pentru depanarea programelor BASIC este utilă trasarea execuției lor prin afișarea numerelor liniilor interpretate, folosind comenzile TRON și TROFF.

Programul introdus de la consolă este păstrat în memoria internă, iar salvarea sa într-un fișier disc se face numai dacă se utilizează comanda SAVE. Există trei forme de memorare pe disc a programelor BASIC: o formă explicită (caractere ASCII) vizibilă prin comenzi CP/M sau cu un editor de texte și două forme incifrate, mai compacte, dar utilizabile numai de către interpretorul BASIC.

Încărcarea de pe disc în memorie a unui program se face prin comanda LOAD, ștergîndu-se din memorie programul executat.

Prin comanda MERGE se pot adăuga programului din memorie alte instrucțiuni citite dintr-un fișier disc, iar prin comanda CHAIN

se poate apela pentru execuție un program de pe disc din programul aflat în memorie.

Din BASIC se poate șterge un fișier disc, prin comanda KILL și se poate schimba numele unui fișier disc prin comanda NAME.

De observat că pentru numele de fișiere literele mici nu sînt transformate automat în litere mari, ca în cazul comenzilor CCP; ca urmare unele fișiere create în BASIC nu pot fi șterse decît din BASIC.

Urmează descrierea comenzilor interpretorului BASIC.

AUTO [nlinie, [, increment]]

Generează automat un număr de linie program după fiecare ⟨CR⟩. Valorile implicite pentru ⟨nlinie⟩ și ⟨increment⟩ sînt de 10. Dacă un număr de linie există deja, după acesta apare „*" ca semn că linia ce urmează va înlocui o linie existentă. Ieșirea din modul „AUTO" se face prin CTRL/C.

CHAIN [MERGE] "fișier" [, [nlinie] [, ALL]
[, DELETE mulțime]]

Apelează un program și îi transmite variabile din programul curent. ⟨fișier⟩ este numele programului apelat, ⟨nlinie⟩ este o expresie care dă numărul liniei de start a programului apelat (implicit prima), „ALL" indică faptul că toate variabilele programului curent vor fi transmise programului apelat (implicit sînt transmise numai cele declarate cu COMMON). Opțiunea „MERGE" adaugă programul apelat la programul din memorie, ca segment separat ce poate fi suprapus peste alt segment. Dacă nu se folosește opțiunea „MERGE", atunci toate declarațiile de variabile din programul inițial se pierd. Opțiunea „DELETE" șterge liniile specificate prin lista ⟨mulțime⟩ și se folosește pentru ștergerea unui segment înainte de a suprapune peste el un alt segment. (⟨mulțime⟩ este de forma ⟨nlinie1⟩-⟨nlinie2⟩). Numerele de linie din ⟨mulțime⟩ sînt afectate de comanda RENUM.

CONT

Continuă execuția unui program după ce aceasta a fost oprită de o comandă CTRL/C sau de o instrucțiune STOP sau END. Se folosește împreună cu STOP pentru depanarea programelor sau pentru reluarea execuției după oprirea prin eroare. CONT este invalid dacă programul a fost schimbat prin editare după oprire.

DELETE [nlinie1] [—] [nlinie2]

Șterge o linie cu numărul ⟨nlinie1⟩, sau toate liniile de program cu numere cuprinse între ⟨nlinie1⟩ și ⟨nlinie2⟩. Dacă ⟨nlinie1⟩ este urmat de „—", atunci se șterge de la ⟨nlinie1⟩ pînă la ultima linie de program; dacă ⟨nlinie2⟩ este precedat de „—", atunci se șterg toate liniile de la prima linie pînă la ⟨nlinie2⟩.

EDIT nlinie

Intră în modul de lucru editare în linia respectivă. Folosind EDIT

se poate modifica o porţiune dintr-o linie fără a reintroduce întreaga linie. Intrarea în mod editare pe linia curentă se poate face şi cu CTRL/A. Subcomenzi de editare:

⟨spaţiu⟩	mută cursorul la dreapta;
⟨RUBOUT⟩	mută cursorul la stînga;
Itext$	inserează text pe poziţia curentă a cursorului;
X	extinde linie; mută cursorul la sfîrşitul liniei, unde se poate insera ⟨text⟩ ⟨CR⟩ (este o inserare la sfîrşit de linie);
nD	şterge n caractere la dreapta cursorului;
H	şterge toate caracterele de la dreapta cursorului;
nSc	caută a n-a apariţie a caracterului ⟨c⟩ şi poziţionează cursorul înainte de el;
nKc	şterge a n-a apariţie a caracterului ⟨c⟩;
Cc	înlocuieşte următorul caracter cu ⟨c⟩;
⟨CR⟩	se salvează modificările făcute în linie şi se iese din modul EDIT; se tipăreşte restul liniei;
E	acelaşi efect ca ⟨CR⟩, însă fără tipărire;
Q	abandon editare linie;
L	listează restul liniei şi poziţionează cursorul la începutul liniei;
EA	permite editarea unei linii din nou, refăcînd linia iniţială şi poziţionînd cursorul pe început de linie.

KILL "fişier"

Şterge fişierul cu numele ⟨fişier⟩ şi de orice tip de pe disc. Fişierul ⟨fişier⟩ nu trebuie să fie deschis.

LIST [nlinie]

LIST [nlinie1] [—] [nlinie2]

Listează o parte sau tot programul aflat în memorie la terminal. Dacă ⟨nlinie⟩ este omis, programul este listat începînd cu linia cu cel mai mic număr şi terminînd cînd se întîlneşte END (sau cînd utilizatorul întrerupe listarea prin CTRL/C). Dacă ⟨nlinie⟩ este prezent, se afişează la terminal numai linia cu acest număr. Al doilea format permite următoarele opţiuni:

— dacă se foloseşte numai ⟨nlinie1⟩ sînt listate toate liniile program cu numărul mai mare sau egal cu acesta;

— dacă se foloseşte numai ⟨nlinie2⟩ se listează toate liniile de la începutul programului pînă la linia respectivă, inclusiv;

— dacă ambele numere de linii sînt specificate, se listează porţiunea de program cuprinsă între ele, inclusiv.

LLIST [nlinie1] [—] [nlinie2]

Listează o parte sau tot programul la imprimantă (se consideră implicit 132 de caractere pe linie la imprimantă).

LOAD "fișier" [, R]
Încarcă un fișier disc de tip „.BAS" în memorie. LOAD închide toate fișierele deschise anterior și șterge toate variabilele și liniile program aflate în memorie înaintea încărcării. Dacă este prezentă opțiunea „R", programul încărcat este și executat, iar toate fișierele de date deschise anterior rămîn neschimbate. LOAD cu opțiunea „R" poate fi folosit pentru a înlănțui mai multe programe (sau segmente ale aceluiași program).

Informațiile pot fi transmise între diferitele programe (sau segmente) folosind fișierele de date pe disc.

MERGE "fișier"
Adaugă fișierul disc specificat la programul curent din memorie. Dacă o linie din fișier are același număr cu o linie din programul rezident în memorie, linia din fișier înlocuiește linia din memorie. După executarea unei instrucțiuni MERGE se intră în modul de lucru direct.

NAME nume2 = nume1
Schimbă numele unui fișier disc. ⟨nume2⟩ trebuie să existe și ⟨nume1⟩ să nu fie prezent pe disc.

NEW
Șterge programul și toate variabilele din memorie. Comanda este folosită pentru a șterge memoria înaintea introducerii unui nou program.

NULL exp
Stabilește numărul de cifre 0 adăugate la sfîrșitul fiecărei linii ca fiind egal cu valoarea expresiei ⟨exp⟩.

RENUM [[nlinie1,[, nlinie2] [, increment]]]
Renumerotarea liniilor program. ⟨nlinie1⟩ este primul număr folosit în noua secvență (implicit este 10). ⟨nlinie2⟩ este numărul liniei cu care începe renumerotarea (implicit prima linie program). ⟨increment⟩ este valoarea cu care se mărește numărul de linie (implicit se consideră 10). RENUM schimbă corespunzător toate argumentele instrucțiunilor GOTO, GOSUB, THEN etc. Nu se poate schimba ordinea liniilor în program.

RESTORE [nlinie]
Permite recitirea unei instrucțiuni DATA. Dacă ⟨nlinie⟩ este prezent, prima instrucțiune READ care urmează instrucțiunii cu numărul ⟨nlinie⟩ va citi prima instrucțiune DATA din program; altfel aceasta va fi citită de prima instrucțiune READ ce urmează lui RESTORE.

RESUME [0] [NEXT] [nlinie]
Continuă execuția după o subrutină de tratare a erorii. În cazul instrucțiunilor RESUME și RESUME 0 execuția programului se reia de la

instrucţiunea care a produs eroarea, pentru RESUME NEXT, de la instrucţiunea următoare acesteia, iar pentru „RESUME nlinie" de la linia specificată.

RUN [nlinie]
Execută un program aflat în memorie începînd cu linia numărul ⟨nlinie⟩ sau cu prima linie de program, dacă ⟨nlinie⟩ lipseşte.

RUN "fişier" [, R]
Încarcă un fişier program de pe disc şi îl execută. RUN închide toate fişierele deschise şi şterge conţinutul memoriei înainte de a încărca un program de pe disc. Dacă opţiunea „R" este prezentă, sînt păstrate toate fişierele de date deschise anterior. ⟨fişier⟩ este numele folosit la salvarea programului pe disc (cu instrucţiunea SAVE).

SAVE "fişier" [, A] [, P]
Salvează un program într-un fişier disc. Dacă există ⟨fişier⟩ pe disc, programul salvat va fi scris peste conţinutul vechiului fişier. Opţiunea „A" determină salvarea programului în format ASCII. Dacă lipseşte „A", atunci BASIC salvează programul într-un alt format diferit de ASCII, mai eficient din punct de vedere al spaţiului pe disc. Opţiunea „P" duce la protejarea programului (orice încercare ulterioară de listare sau de editare a programului se va solda cu un mesaj de eroare). Pentru a putea folosi ulterior instrucţiunea MERGE, programul trebuie salvat în format ASCII.

TRON
TROFF
Activează (TRON) sau dezactivează (TROFF) trasarea execuţiei unui program BASIC prin afişarea numerelor liniilor executate, între paranteze drepte.

11.2. Limbajul BASIC-80

11.2.1. Elemente de bază ale limbajului

O linie program poate avea următoarea formă:
nnnnn instrucţiune [: instrucţiune ...]
Rezultă că se pot introduce mai multe instrucţiuni pe o linie program separate prin „:". O linie program poate avea cel mult 255 caractere şi se poate extinde pe mai multe linii fizice; în acest caz fiecare linie fizică este terminată cu ⟨LF⟩.

Fiecare instrucţiune începe cu un *număr de linie* ⟨nnnnn⟩. Acest număr arată ordinea în care liniile program vor fi memorate şi este folosit ca referinţă în instrucţiunile de salt şi de editare. Numărul liniei este cuprins între 0 şi 65529. Caracterul „." poate fi folosit

în comenzile EDIT, LIST, AUTO și DELETE pentru referire la linia curentă.

Setul de caractere cuprinde caractere alfabetice, caractere numerice și caractere speciale. Caracterele alfabetice reprezintă literele mari și mici ale alfabetului latin. Caracterele numerice sînt cifrele de la 0 la 9.

Se pot utiliza atît litere mari, cît și litere mici, deoarece ele sînt echivalente; în cadrul constantelor de tip șir, inclusiv cele care reprezintă nume de fișiere, se face diferență între literele mici și literele mari.

Caracterele speciale recunoscute de BASIC sînt:

= + — * / ^ ( ) % # $! [ ] , ; : ' . & ? ⟨ ⟩ / @ —
⟨RUBOUT⟩ ⟨ESC⟩ ⟨LF⟩ ⟨CR⟩

Constantele BASIC pot fi de două tipuri: numerice și șiruri.

O constantă de tip șir este o secvență de pînă la 255 caractere alfanumerice, cuprinse între ghilimele (duble). Exemplu: "JOC.BAS".

Constantele numerice sînt numere pozitive și negative. BASIC recunoaște 5 tipuri de constante numerice:

— constante întregi, zecimale, cuprinse în intervalul — 32768 ÷ 32767 (fără punct zecimal);
— constante neîntregi cu punct zecimal: 452.32;
— constante neîntregi cu exponent: 235E5;
— constante în dublă precizie: 325D5;
— constante hexazecimale, precedate de &H:&HC3;
— constante octale, precedate de &0:&0777.

Constantele pot fi reprezentate în simplă precizie (memorate cu 7 cifre și reprezentate cu maxim 6 cifre) și dublă precizie (cu 16 cifre). O constantă în precizie simplă este recunoscută prin una din următoarele caracteristici:

— are maximum 7 cifre zecimale;
— are forma cu exponent (cu litera E);
— se termină cu caracterul !.

O constantă în dublă precizie este recunoscută prin una din următoarele caracteristici:

— are peste 8 cifre zecimale;
— are o formă exponențială (cu litera D);
— se termină cu caracterul #.

Variabilele BASIC pot să aibe nume de orice lungime, dar numai primele 40 de caractere sînt semnificative.

Numele unei variabile poate conține litere, cifre și caracterul punct („.") și trebuie să nu coincidă cu un nume rezervat; numele rezervate sînt numele de comenzi, de instrucțiuni, de funcții și de operatori (logici).

Dacă un nume începe cu literele FN, atunci el este un nume de funcție definită de utilizator.

Variabilele pot fi variabile șir, întregi, simplă precizie sau dublă precizie.

Tipul unei variabile rezultă implicit din numele variabilei, mai precis dintr-un caracter special ce urmează numele variabilei. Acest caracter poate fi:

$ — variabilă de tipul șir;

% — variabilă întreagă;

! — variabilă simplă precizie;

—variabilă dublă precizie.

Dacă numele unei variabile nu se termină cu nici unul dintre aceste caractere, atunci tipul ei implicit este simplă precizie.

Este posibilă și declararea explicită a tipului variabilelor al căror nume începe cu anumite litere prin instrucțiunile: DEFINT, DEFSTR, DEFSNG, DEFDBL.

BASIC acceptă și variabile indexate. Numărul maxim de dimensiuni este 255. Numărul maxim de elemente pe dimensiune este 32767.

O variabilă indexată se declară folosind instrucțiunea DIM.

Numele unei variabile indexate este orice nume de variabilă acceptat de BASIC și este urmat de una sau mai multe expresii întregi între paranteze. De exemplu: A(1, 2), B(i, j), C(k).

Alocarea de memorie pentru variabile se face astfel:
pentru variabile numerice:

— întregi: 2 octeți;

— simplă precizie: 4 octeți;

— dublă precizie: 8 octeți;

pentru variabile șir:

— 3 octeți în plus față de numărul caracterelor din șir.

Conversii de tip:

– dacă se atribuie unei variabile numerice o constantă de un tip diferit de al variabilei, atunci valoarea numerică se convertește automat la tipul variabilei;

— la evaluarea expresiilor, toți operanzii sînt convertiți automat la tipul operandului cu precizia cea mai mare, care este și tipul rezultatului expresiei;

— operatorii logici convertesc operanzii în întregi și produc un rezultat întreg;

— la conversia din virgulă mobilă în întreg, se face rotunjire la întregul cel mai apropiat și nu se trunchiază partea fracționară;

— la conversia din dublă precizie în simplă precizie se rețin numai primele 7 cifre zecimale, cu rotunjire la ultima cifră.

O expresie poate fi o constantă, o variabilă sau o succesiune de constante și variabile legate prin operatori și eventual cuprinse între paranteze rotunde.

Există 4 categorii de operatori:
— aritmetici;
— relaționali;
— logici;
— funcționali.

Operatorii aritmetici, în ordinea crescătoare a priorității, sînt următorii:

adunarea (+) și scăderea (—)
înmulțirea (×), împărțirea (/), împărțirea întreagă (\) și restul împărțirii întregi (MOD),
ridicarea la putere (^)

Ordinea executării operațiilor poate fi modificată folosind parantezele „(" și „)".

Operatorii aritmetici (\) și MOD se pot folosi și cu operanzi neîntregi, care sînt convertiți la întregi înainte de operație.

În cazul unei depășiri sau împărțiri cu 0 se afișează un mesaj, se pune ca rezultat numărul maxim reprezentabil și se continuă execuția programului.

Operatorii relaționali permit compararea a două valori și sînt folosiți pentru a compara expresii aritmetice sau șiruri:

=, <, >, <= sau =<, >= sau =>, < > sau > < (diferit)

Rezultatul unei operații relaționale poate fi „adevărat" (valoare = = —1) sau „fals" (valoare = 0).

Operațiile aritmetice sînt evaluate înaintea operațiilor de relație.

Operatorii logici, în ordinea descrescătoare a priorității:

NOT — negație;
AND — și logic;
OR — sau logic;
XOR — sau exclusiv;
IMP — implicație (X IMP Y = 1, dacă X <= Y);
EQV — echivalență (X EQV Y = 1, dacă X = Y).

Rezultatul unei operații logice este fie „adevărat" (valoare diferită de 0), fie „fals" (0).

Operațiile logice se fac bit cu bit.

Operațiile permise asupra șirurilor de caractere sînt concatenarea (prin operatorul „+") și comparația (prin operatorii de relație).

În BASIC pot fi folosite două tipuri de funcții: funcții intrinseci (predefinite) și funcții definite de utilizator (prin instrucțiunea DEFFN).

11.2.2. Instrucțiuni de prelucrare și de control

CALL var [(var,...)]

Se apelează o subrutină în limbaj de asamblare și eventual se transmit argumentele necesare. Variabila care reprezintă numele subrutinei trebuie să aibă ca valoare adresa de start a subrutinei. Instrucțiunea

CALL generează aceeaşi secvenţă de apelare ca în FORTRAN F80 şi COBOL.

CLEAR [, [exp1] [, expr2]]
Iniţializează toate variabilele numerice cu 0, toate variabilele de tip şir la valoarea " " (şir nul) şi închide toate fişierele deschise. ⟨exp1⟩ este o locaţie de memorie care, dacă este specificată, reprezintă limita superioară a memoriei disponibile pentru interpretor. ⟨exp2⟩ stabileşte noua dimensiune pentru stiva BASIC (dimensiunea implicită a stivei este de 256 octeţi). Exemple:

CLEAR	toate variabilele iau valoarea 0 şi se închid toate fişierele;
CLEAR, 32768	fixează limita memoriei disponibile pentru interpretor la adresa 32768;
CLEAR, ,2000	rezervă pentru stivă 2000 de octeţi.

COMMON var[,var,. . .,
Transmite implicit variabilele din ⟨lista variabile⟩ unui program la execuţia unei instrucţiuni CHAIN. Variabilele de tip tablou trebuie să fie însoţite de paranteze (ex: COMMON A, B ( )). Este posibilă şi o instrucţiune COMMON similară cu cea din FORTRAN:

COMMON /nume/ var[,var,. . .]
pentru comunicare cu programe în FORTRAN sau în limbaj de asamblare, dar nu pentru transmiterea variabilelor la un program BASIC apelat cu CHAIN. Exemplu:

```
100 COMMON A, B, C, D$
110 CHAIN "PROG3", 10
```

DATA const[,const,. . .]
Memorează constante numerice şi şiruri de caractere care pot fi citite ulterior cu instrucţiunea READ. Poate fi plasată oriunde în program, deoarece instrucţiunea READ accesează instrucţiunile DATA în ordine. La citire, tipul variabilei trebuie să corespundă cu cel stabilit prin DATA. Citirea instrucţiunilor DATA poate fi reluată folosind instrucţiunea RESTORE.

DEF FNnume [(var,. . .)] = exp
Defineşte o funcţie utilizator şi parametrii săi. ⟨nume⟩ trebuie să respecte convenţiile unui nume de variabilă. Definirea unei funcţii trebuie făcută înaintea utilizării ei. O funcţie poate fi de tip numeric sau şir. Definirea nu este posibilă în modul de lucru direct.

DEFtip litera[—litera]
unde ⟨tip⟩ poate fi: INT, SNG, DBL sau STR. Declară tipul variabilelor ale căror nume încep cu una dintre literele specificate ca întreg, simplă precizie, dublă precizie, sau şir. Implicit toate variabilele sînt considerate de tip simplă precizie. Exemplu: DEFINT I—N: DEFSTR A

DEF USR[cifra] = exp
Specifică adresa de început a unei funcții în limbaj de asamblare printr-o expresie întreagă ⟨exp⟩. ⟨cifra⟩ este între 0 și 9 și corespunde numărului de funcție a cărei adresă este dată. Dacă este omis, rezultă că se specifică adresa subrutinei 0 (DEF USR0). Într-un program pot apărea mai multe DEF USR pentru a redefini punctul de start al unei funcții, ceea ce permite un număr nelimitat de subrutine în limbaj de asamblare.

DIM var(const[,const,...])
Specifică dimensiunile maxime ale variabilelor indexate. Dacă dimensiunea unei variabile indexate nu este specificată, este stabilită implicit la 10. Dacă dimensiunea maximă este depășită apare un mesaj de eroare. Instrucțiunea DIM inițializează toate componentele cu zero.

END
Termină execuția programului, închide toate fișierele și dă controlul interpretorului BASIC. Instrucțiunea END poate fi plasată oriunde în program pentru a termina execuția. Spre deosebire de STOP, nu determină apariția unui mesaj la consolă.

ERASE var[,var,...]
Elimină variabilele indexate din program. Zona de memorie eliberată poate fi folosită pentru redimensionarea variabilelor indexate sau în alte scopuri.

ERR și **ERL**
Variabilele ERR și ERL sînt folosite în caz de eroare astfel: cînd apare o eroare, ERR conține un cod de eroare și ERL numărul liniei program la care s-a produs eroarea. Pot fi folosite în instrucțiuni de tip IF. Dacă eroarea a apărut în modul de lucru direct, ERL conține 65535. ERL și ERR nu pot apărea în instrucțiuni de atribuire, deoarece sînt variabile rezervate.

ERROR exp
Simulează apariția unei erori BASIC și permite definirea unor coduri de eroare de către utilizator. Valoarea expresiei (între 0 și 255) reprezintă un cod de eroare BASIC. La execuția instrucțiunii ERROR apare mesajul de eroare corespunzător. Pentru a defini un cod de eroare propriu, utilizatorul trebuie să folosească o valoare mai mare decît cele folosite pentru coduri de eroare de BASIC. Acest cod de eroare utilizator trebuie să ducă la execuția unei rutine de eroare.

FOR var = expi TO expf [STEP exps]
...
NEXT [var] [,var,...]
Permite un ciclu de mai multe instrucțiuni. Variabila ⟨var⟩ este variabilă contor pentru ciclu, ⟨expi⟩ este valoarea sa inițială, ⟨expf⟩

valoarea sa finală, iar ⟨exps⟩ incrementul (dacă lipsește se ia implicit 1). Pasul poate fi și negativ. Ciclurile FOR pot fi incluse, dar trebuie să aibă variabile de control diferite. O instrucțiune NEXT fără variabilă este asociată cu ultima instrucțiune FOR.

GOSUB nlinie

Apelează subrutina Basic care începe cu linia ⟨nlinie⟩.

GOTO nlinie

Salt necondiționat la linia cu numărul specificat, sau la prima instrucțiune executabilă după linia specificată.

IF exp **THEN** instruct [ELSE instruct]
IF exp **THEN** nlinie [ELSE nlinie]
IF exp **GOTO** nlinie [ELSE nlinie]

Testează valoarea expresiei ⟨exp⟩ și: dacă ⟨exp⟩ ⟨ ⟩ 0, se execută ramura lui THEN sau GOTO; dacă ⟨exp⟩ = 0, se execută ramura lui ELSE dacă există. Execuția continuă cu următoarea instrucțiune. Este posibil ca o instrucțiune IF să includă o altă instrucțiune IF. Dacă o instrucțiune IF...THEN este urmată de un număr de linie în modul de lucru direct, atunci apare o eroare. Cînd se folosește IF pentru a testa mărimea rezultatului unei operații cu numere în virgulă mobilă, trebuie să se țină seama de faptul că valoarea poate să nu fie exactă.

[**LET**] var = exp

Se atribuie unei variabile rezultatul evaluării expresiei ⟨exp⟩. Cuvîntul cheie LET poate lipsi.

MID$ (⟨expsir1⟩, n[,m] = ⟨expsir2⟩

unde n și m sînt expresii întregi și ⟨expsir⟩ este o expresie de tip șir (o variabilă sau o constantă șir). Înlocuiește caracterele din ⟨expsir1⟩, începînd cu poziția n, cu cele din ⟨expsir2⟩ (sau cu m caractere din ⟨expsir2⟩ dacă m este prezent).

ON ERROR GOTO nlinie

Salt în caz de eroare la rutina de tratare a erorii scrisă de utilizator. O instrucțiune ON ERROR GOTO 0 sare la rutinele de tratare a erorilor din interpretor, care tipăresc un mesaj de eroare și opresc execuția. Dacă apare o eroare pe timpul execuției unei rutine de tratare a erorii, se afișează un mesaj de eroare BASIC și se oprește execuția programului.

ON exp **GOSUB** nlinie[,nlinie,..]
ON exp **GOTO** nlinie [,nlinie,...]

Salt la una dintre diferitele linii specificate în listă, condiționat de valoarea expresiei în momentul executării instrucțiunii. De exemplu, dacă, în urma evaluării, expresia ia valoarea 3, atunci saltul se va face la linia al cărui număr ocupă poziția 3 în listă (dacă valoarea nu este

întreagă se rotunjește). Dacă valoarea expresiei este 0 sau mai mare decît numărul de elemente din listă, se continuă cu următoarea linie executabilă a programului. Numerele de linii folosite în GOSUB trebuie să apară la început de subrutine. Dacă valoarea expresiei este negativă, apare un mesaj de eroare.

OPTION BASE n

Declară valoarea minimă pentru indicii variabilelor indexate. Valoarea implicită este 0.

POKE adr, exp

Scrie octetul ⟨exp⟩ în locația de memorie cu adresa ⟨adr⟩. Funcția complementară lui POKE este PEEK care citește conținutul unei locații de memorie. POKE și PEEK sînt utile pentru memorarea eficientă a datelor, pentru încărcarea de subrutine în limbaj de asamblare, pentru transmiterea de argumente și rezultate la și de la subrutine în limbaj de asamblare.

RANDOMIZE exp

Inițializează generatorul de numere aleatorii. Dacă ⟨exp⟩ lipsește, interpretorul așteaptă o valoare numerică de la consolă înainte de a executa RANDOMIZE. Dacă generatorul de numere aleatorii nu este inițializat, funcția RND va da aceeași secvență de numere aleatorii la fiecare execuție a unui program. Deci este bine să se pună o instrucțiune RANDOMIZE la începutul unui program ce folosește numere aleatoare.

REM comentariu

Permite inserarea unor linii cu comentarii în program. Se mai pot introduce și comentarii precedate de un apostrof oriunde în textul sursă. Comentariile din instrucțiuni DATA vor fi considerate ca date.

STOP

Termină execuția programului și redă controlul interpretorului BASIC. Instrucțiunea STOP nu determină închiderea fișierelor. Cînd se întîlnește un STOP în program se afișează un mesaj.

SWAP var1, var2

Schimbă între ele valorile a două variabile. Cele două variabile trebuie să aibă același tip (întreg, simplă precizie, dublă precizie, șir).

WHILE exp

...

instrucțiuni

...

WEND

Se execută repetat o secvență de instrucțiuni cît timp condiția este adevărată, adică expresia diferită de zero (ciclu cu testul la început). Pot exista oricîte nivele de cicluri de acest gen. Fiecare WEND închide cel mai apropiat WHILE.

11.2.3. Instrucţiuni de intrare-ieşire

CLOSE[#] nfişier[,[#] nfişier...]]
Închide unul sau mai multe fişiere pe disc. ⟨nfişier⟩ este numărul sub care fişierul a fost deschis. O instrucţiune CLOSE fără argumente închide toate fişierele deschise. Fişierul poate fi deschis sub acelaşi număr sau sub alt număr. Comenzile END şi NEW închid automat toate fişierele (STOP nu închide fişiere).

FIELD[#] nfişier [,dim AS var[,var,...],...]
Alocă variabilele specificate în zona buffer a unui fişier în acces direct. ⟨nfişier⟩ este numărul fixat pentru fişier la deschidere. ⟨dim⟩ este numărul de octeţi necesari pentru variabile. Nu se foloseşte pentru variabile din LET sau INPUT. Numărul total al octeţilor alocaţi într-o instrucţiune FIELD nu trebuie să depăşească lungimea înregistrării specificate la deschiderea fişierului în acces direct (lungimea implicită este de 128 octeţi).

Se pot executa mai multe instrucţiuni FIELD pentru acelaşi fişier. Execuţia unei instrucţiuni FIELD nu anulează efectul unei alte instrucţiuni FIELD. Exemplu:

FIELD 1, 20 AS N$, 10 AS ID$.

GET[#] nfişier[,ninreg]
Citeşte înregistrarea cu numărul ⟨ninreg⟩ dintr-un fişier disc într-un buffer în acces direct. ⟨nfişier⟩ este numărul ce a fost atribuit fişierului la deschidere. Dacă lipseşte ⟨ninreg⟩, atunci se citeşte următoarea înregistrare din fişier. Cel mai mare număr de înregistrare posibil este 32767. După o instrucţiune GET se pot citi caractere din buffer cu INPUT# sau LINE INPUT#.

INPUT [;] ["text" ;] var [, var,..]
Permite introducerea de date de la terminal pe timpul execuţiei unui program. Cînd se întîlneşte o instrucţiune INPUT, execuţia programului se întrerupe şi apare pe ecran un mesaj care anunţă utilizatorul că se aşteaptă date, urmat de „?" şi precedat opţional de şirul „text". Pentru a suprima caracterul „?" se poate folosi „ ," după ⟨text⟩ sau „;" după „INPUT".

Datele introduse sînt atribuite în ordine variabilelor din listă. Execuţia programului este reluată abia după ce s-au introdus valori pentru toate variabilele din listă. Dacă se introduc mai multe sau mai puţin date sau de tip necorespunzător apare mesajul: „?Redo from start". Nu se face atribuirea pînă la introducerea corectă.

INPUT # ⟨nfişier⟩, var [,var,...]
Citeşte date dintr-un fişier secvenţial cu numărul ⟨nfişier⟩ şi le atribuie în ordine variabilelor din listă. Primul caracter din fişier diferit de ⟨CR⟩, spaţiu sau ⟨LF⟩ reprezintă începutul unui număr. Sfîrşitul

unui număr este considerat caracterul ce precede un spațiu, ⟨CR⟩, sau „,". Dacă se întîlnește caracterul ghilimele ("), se citește un șir delimitat de alt caracter ghilimele sau de sfîrșitul fișierului.

LINE INPUT[;] ["text";] var
Citește de la consolă o linie terminată cu ⟨CR⟩ (pînă la 255 caractere) într-o variabilă de tip șir ⟨var⟩. ⟨text⟩ este un șir de caractere care, dacă este prezent, este afișat înainte ca intrarea să fie acceptată. Dacă LINE INPUT este urmat de „;", atunci un ⟨CR⟩ tastat de utilizator nu va avea ca ecou ⟨CR⟩ ⟨LF⟩ la consolă, așa cum se întîmplă cînd lipsește acest caracter. Ieșirea din execuția unei instrucțiuni LINE INPUT se face cu ^C.

LINE INPUT #nfișier, var
Citește o linie terminată cu ⟨CR⟩ (fără delimitatorii ⟨CR⟩⟨LF⟩), dintr-un fișier disc secvențial ⟨nfișier⟩ într-o variabilă șir ⟨var⟩.

LPRINT [exp[,exp,...]]
LPRINT USING expsir; exp [exp,...]
Tipărește valorile expresiilor din listă la imprimantă. A se vedea și instrucțiunea PRINT pentru opțiunea USING.

LSET var[,var...] = exp[,exp,..]
RSET var[,var..] = exp[,exp,..]
Mută date din memorie într-un buffer de fișier în acces direct (pentru pregătirea unei instrucțiuni PUT). Dacă șirul de expresii necesită mai puțini octeți decît au fost rezervați cu FIELD pentru lista de variabile, LSET aliniază șirul la stînga, iar RSET aliniază șirul la dreapta. Restul cîmpului se completează cu spații. Un șir prea lung de caractere este trunchiat la dreapta. Valorile numerice trebuie convertite în șiruri înainte de a fi transferate. LSET și RSET pot fi folosite și pentru alinierea unor șiruri de caractere la stînga, respectiv la dreapta. Exemplu:

```
110 A$ = SPACE$(20)
120 RSET A$ = N$
```

aliniază la dreapta șirul N$ într-un cîmp de 20 de caractere.

OPEN "mod", [#] nfișier, "fișier", [lungime]
Deschide un fișier pe disc în vederea creării sau exploatării sale. OPEN alocă un buffer pentru operațiile de I/E și determină modul de acces. ⟨mod⟩ este un șir de expresii în care primul caracter va fi unul dintre următoarele:

O — specifică mod de ieșire secvențial;
I — specifică mod de intrare secvențial;
R — specifică mod I/E în acces direct.

⟨nfișier⟩ este o expresie întreagă cu valori de la 1 la 15. Numărul ⟨nfișier⟩ este asociat fișierului pe toată durata cît acesta este deschis

şi este folosit în instrucţiunile de I/E. <lungime> este o expresie care, dacă este prezentă, stabileşte lungimea înregistrării pentru operaţii I/E în acces direct (implicit lungimea unei înregistrări este 128 octeţi).

Un acelaşi fişier poate fi deschis sub mai multe numere pentru citire secvenţială sau acces direct, dar numai cu un singur număr pentru scriere secvenţială.

OUT port,exp

Trimite valoarea expresiei <exp> pe un octet la portul de ieşire <port>.

PRINT [exp [,exp,...]]
PRINT [exp [;exp;...]]

Afişează la consolă valorile expresiilor din listă. Dacă nu există nici o expresie, atunci se trece la linia următoare. Expresiile din listă pot fi numerice şi (sau) şiruri de caractere. Poziţia fiecărui element tipărit este determinată de caracterele folosite pentru separarea elementelor în lista de expresii. Linia de afişare este împărţită în zone de cîte 14 poziţii fiecare. O virgulă („,") între două elemente ale listei determină afişarea valorii următoare în următoarea zonă. „;" determină afişarea valorii următoare imediat după cea anterioară. Unul sau mai multe spaţii între expresii au acelaşi efect cu caracterul „;".

Dacă lista de expresii se termină cu „," sau „;", atunci următoarea instrucţiune PRINT începe afişarea în continuare, iar în caz contrar următoarea afişare începe pe o linie nouă. Dacă se ajunge la sfîrşitul unei linii fizice în timpul unei afişări, se continuă pe linia următoare. Numerele afişate sînt urmate de un spaţiu, iar numerele pozitive sînt precedate de un spaţiu.

PRINT USING expsir; exp[,exp,..]

Afişează valorile unei liste de expresii şir sau numerice, conform formatului specificat în <expsir>. Pentru afişarea de şiruri <expsir> poate conţine următoarele caractere speciale:

"!"

Se afişează numai primul caracter din şir.

„\n spaţii\"

Se afişează 2+n caractere din şir. Dacă n=0, se vor afişa 2 caractere din şir; dacă şirul este mai lung decît cîmpul rezervat pentru afişare, caracterele de la sfîrşit sînt ignorate; dacă spaţiul pentru afişare este mai mare decît şirul, acesta se va completa cu spaţii la dreapta.

„&"

Specifică un cîmp rezervat pentru afişarea unui şir de lungime variabilă. Şirul este afişat în întregime, iar între două şiruri nu apar spaţii decît dacă ele fac parte dintr-unul din şiruri. Pentru afişarea de numere <expsir) poate conţine următoarele caractere speciale:

„#"

Folosit pentru a reprezenta poziţia fiecărei cifre a numărului. Secvenţa

de „#" trebuie să fie urmată de numărul dorit de spaţii. La reprezentarea numerelor zecimale cu acest format se vor face rotunjirile corespunzătoare. Exemplu

PRINT USING "##.##"; 5.3,6.789 produce 5.30 6.79.

„+"

Aflat la începutul sau la sfîrşitul şirului format determină afişarea semnelor numerelor (+ sau —) înaintea, respectiv în urma lor.

„—"

Pus la sfîrşitul formatului determină afişarea semnului pentru numere negative.

„**"

Puse la începutul formatului determină apariţia unor asteriscuri înaintea unor numere şi oferă posibilitatea afişării unor numere cu două cifre mai lungi decît formatul specificat. Exemplu:

PRINT USING "**#. #"; 12.39,0.9,765.1 produce *12.4 ** 0.9765.1

„$$"

Duce la apariţia unui „$" imediat la stînga numărului. Creează posibilitatea afişării unei cifre în plus faţă de format.

„**$"

Aflat la începutul formatului combină efectele celor două anterioare. Exemplu:

PRINT USING "**$##.##"; 2.34 produce ***$2.34.

„,"

O virgulă aflată în format la stînga punctului zecimal determină apariţia unei virgule la fiecare 3 cifre în stînga punctului zecimal. O virgulă aflată la sfîrşitul formatului apare după fiecare număr afişat. Virgula nu are efect, cînd se utilizează formatul exponenţial (^^^^)

„^^^^"

Plasate după poziţiile cifrelor, specifică formatul exponenţial. Exemplu:

PRINT USING "####^^^^"; 888888 produce .8889E06.

„\"

Următorul caracter din format se afişează. Exemplu:

PRINT USING "\!##. ##\!"; 11.22 produce !11.22!.

„%"

Dacă numărul de afişat este mai mare (sau devine mai mare prin rotunjire decît cîmpul specificat prin format apare „%" înaintea numărului. Exemplu:

PRINT USING ".##";.999 produce %1

PRINT USING "##.##"; 111.22 produce %111.22.

Dacă numărul de cifre specificat este mai mare decît 24 apare un mesaj de eroare.

PRINT # nfişier [USING expsir;] exp[,exp,...]

Scrie date într-un fişier secvenţial. ⟨nfişier⟩ este numărul folosit la deschiderea fişierului. ⟨expsir⟩ este şirul format, prezentat la instruc-

țiunea PRINT. PRINT # nu compactează datele, ci le scrie așa cum ar fi afișate de PRINT. Pentru a obține un cîmp compact de date în fișier se utilizează „;" ca delimitator în lista de expresii. Dacă se folosește „," (virgula), se scriu pe disc și spațiile care ar apărea la afișare.

PUT[#] nfișier [,ninreg.]
Scrie o înregistrare din buffer într-un fișier deschis în acces direct cu numărul ⟨nfișier⟩. Dacă ⟨ninreg.⟩ lipsește atunci înregistrarea va primi numărul următor celui dat ultimei înregistrări în fișier (la ultima folosire a instrucțiunii PUT). Cel mai mic număr de înregistrare este 1 și cel mai mare 32767. Instrucțiunile PRINT #, PRINT # USING, WRITE # pot fi utilizate pentru a pune caractere într-un buffer de fișier disc în acces direct înaintea unei instrucțiuni PUT. În cazul instrucțiunii WRITE #, zona buffer va fi umplută cu spații după ⟨CR⟩. Orice încercare de a citi sau scrie după sfîrșitul zonei buffer duce la o eroare.

READ var[,var,..]
Atribuie variabilelor din listă valori din instrucțiuni DATA. O instrucțiune READ poate folosi una sau mai multe instrucțiuni DATA și diferite instrucțiuni READ pot folosi aceeași instrucțiune DATA, folosind instrucțiunea RESTORE. Dacă numărul variabilelor din listă depășește numărul de elemente dintr-o instrucțiune DATA, apare un mesaj „out of DATA". Dacă numărul de variabile specificat este mai mic decît numărul de elemente, următoarea instrucțiune READ va începe citirea cu primul element necitit din instrucțiunea DATA. Dacă se termină instrucțiunile READ, datele necitite sînt ignorate. Pentru a reciti valori din DATA se folosește RESTORE.

WAIT port,i,[j]
Suspendă execuția programului pînă cînd portul specificat conține un anumit octet. Octetul citit de la portul ⟨port⟩ intră într-o expresie SAU LOGIC cu expresia întreagă ⟨j⟩ (dacă există) și o expresie ȘI LOGIC cu expresia întreagă ⟨i⟩. Dacă rezultatul produsului logic este 0 portul este recitit în ciclu; dacă este diferit de zero se trece la instrucțiunea următoare.

WIDTH [LPRINT] exp
Stabilește mărimea liniei (în număr de caractere) pentru consolă sau pentru imprimantă (opțiunea LPRINT). Expresia întreagă ⟨exp⟩ trebuie să fie între 15 și 255. Mărimea implicită este de 72 de caractere. Dacă ⟨exp⟩ are valoarea 255 mărimea liniei este „infinită", adică interpretorul nu introduce un caracter ⟨CR⟩ .

WRITE [exp[,exp,..]]
Afișează date la terminal. Dacă lista de expresii lipsește, se afișează ⟨CR⟩⟨LF⟩. După ultimul element din listă, BASIC inserează ⟨CR⟩ ⟨LF⟩.

WRITE# nfişier, exp[,exp,...]
Scrie date într-un fişier secvenţial. Diferenţa dintre WRITE# şi PRINT# constă în faptul că WRITE# inserează o virgulă după fiecare element introdus în fişier şi ⟨CR⟩ după ultimul element din listă. ⟨nfişier⟩ este numărul sub care fişierul a fost deschis în modul „O". Expresiile pot fi numerice sau de tipul şir.

11.3. Funcţii, subrutine şi comenzi grafice

11.3.1. Funcţii predefinite în BASIC-80

Aceste funcţii pot fi apelate în orice program fără a fi definite înainte. În descrierea formatului funcţiilor argumentele vor avea următoarele semnificaţii:

x, y,—orice expresii numerice;
i, j,—expresii cu rezultat întreg;
x$, y$,—expresii şir.

Dacă se foloseşte un număr zecimal în locul unui număr întreg, numărul zecimal va fi rotunjit.

Funcţiile interpretorului BASIC—80 au ca rezultate numai valori întregi şi în simplă precizie.

Lista funcţiilor BASIC.

ABS(x)
Produce valoarea absolută a expresiei x.

ASC(x$)
Produce codul ASCII al primului caracter din şirul x$. Dacă x$ este nul, se dă un mesaj de eroare.

ATN(x)
Produce valoarea arctg(x) în radiani. Rezultatul este cuprins între —PI/2 şi PI/2 şi este în simplă precizie.

CDBL(x)
Converteşte o valoare x în dublă precizie.

CHR$(i)
Produce caracterul cu codul ASCII „i". Se poate folosi în comanda PRINT, pentru a trimite caractere de control la consolă.

CINT(x)
Converteşte valoarea x la un număr întreg prin rotunjirea părţii fracţionare.

COS(x)
Produce valoarea cos(x), unde x este în radiani.

CSNG(x)

Converteşte pe x într-un număr în simplă precizie.

CVI(expsir2)
CVS(expsir4)
CVD(expsir8)

Converteşte un şir de octeţi în valoare numerică. CVI converteşte un şir de 2 caractere la un întreg, CVS un şir de 4 caractere la o valoare în simplă precizie, iar CVD un şir de 8 octeţi la o valoare în dublă precizie.

Şirurile numerice citite dintr-un fişier în acces direct trebuie convertite în valori numerice înaintea folosirii lor în expresii aritmetice.

EOF(nfişier)

Produce — 1 (adevărat) dacă s-a ajuns la sfîrşitul unui fişier secvenţial, identificat prin numărul ⟨nfişier⟩. Se foloseşte la citirea datelor dintr-un fişier pentru a evita apariţia unei erori la încercarea de a citi după sfîrşitul de fişier.

EXP(x)

Produce valoarea exponenţialei. Dacă x este mai mare decît 87.3365, apare un mesaj de depăşire şi execuţia programului continuă.

FIX(x)

Trunchiază valoarea x şi produce partea sa întreagă.

FRE(0)
FRE(x$)

Produce numărul octeţilor de memorie nefolosiţi încă de interpretor. Argumentul x$ nu este utilizat de funcţie. FRE(" ") activează o „colectare" a octeţilor nefolosiţi înainte de a da numărul lor.

HEX$(x)

Produce valoarea hexazecimală a unui întreg. Dacă x nu este întreg, atunci el va fi rotunjit înainte de a se calcula valoarea hexazecimală.

INKEY$

Are ca rezultat un caracter citit de la consolă sau un şir nul, dacă nu a fost introdus nici un caracter de la terminal. Caracterul primit de INKEY nu este afişat în ecou.

INP(i)

Are ca rezultat octetul citit la portul i (i între 0 şi 255). INP este funcţia complementară instrucţiunii OUT.

INPUT$(x[,[#]i])

Produce un şir de x caractere citite de la terminal sau din fişierul cu numărul „i" (dacă „i" este specificat). Caracterele citite nu vor fi afişate.

INSTR([i,]x$, y$)
Caută prima apariție a șirului y$ în șirul x$ și întoarce poziția de la care începe corespondența. Argumentul opțional i indică poziția de la care se începe căutarea. Dacă i> LEN(x$) sau x$ este nul sau y$ nu poate fi regăsit în x$, funcția întoarce valoarea 0. Dacă y$ este șir nul funcția are ca rezultat i (dacă există), sau 1.

INT(x)
Produce partea întreagă a lui x.

LEFT$(x$, i)
Are ca rezultat un subșir al lui x$ cuprins între pozițiile 1 și i inclusiv. Dacă i>LEN(x$), se obține întregul șir; dacă i = 0, se obține șirul nul.

LEN(x$)
Produce lungimea șirului x$. Sînt numărate și caracterele neafișabile și spațiile.

LOC(nfișier)
Produce numărul ultimei înregistrări citite sau scrise într-un fișier disc deschis în acces direct. Dacă fișierul a fost deschis dar nu a avut loc nici o operație de I/E, LOC întoarce valoarea 0. Pentru fișierele disc secvențiale LOC întoarce numărul de sectoare care au fost citite sau scrise.

LOG(x)
Produce logaritmul natural al lui x (x > 0).

LPOS(x)
Are ca rezultat poziția următorului caracter din zona buffer de afișare, x este neutilizat și poate avea orice valoare.

MID$ (x$,i[, j])
Produce un subșir de j caractere din x$, care încep cu poziția i. Dacă j este omis sau j > LEN(x$)—i, atunci se obține subșirul de lungime LEN(x$)—i + 1. Dacă i > LEN(x$), MID$ întoarce un șir nul. Dacă i = 0, apare un mesaj de eroare.

MKI$(exp. întreagă)
MKS$(exp. simplă precizie)
MKD$(exp. dublă precizie)
Aceste funcții convertesc valori numerice în șiruri de valori. MKI$ convertește un întreg într-un șir de 2 octeți, MKS$ convertește un număr simplă precizie într-un șir de 4 octeți, iar MKD$ un număr dublă precizie într-un șir de 8 octeți. Orice valoare plasată într-un buffer în acces direct cu instrucțiunile LSET sau RSET trebuie convertită în șir de valori.

OCT$(x)
Produce valoarea octală a unui număr întreg. x este rotunjit la valoarea întreagă înaintea conversiei.

PEEK(i)

Produce valoarea întreagă a octetului citit din memorie în locația i. PEEK este funcția complementară lui POKE.

POS(i)

Are ca rezultat poziția curentă a cursorului pe linie (valoarea minimă este 1).

RIGHT$(x$, i)

Produce un subșir al lui x$ ce cuprinde caracterele de la începutul lui x$ pînă la poziția i. Dacă i >= LEN(x$), atunci se obține tot șirul x$. Dacă i = 0, se obține un șir nul.

RND[(x)]

Produce o valoare aleatorie în intervalul [0, 1]. Pe parcursul mai multor execuții ale aceluiași program se va obține aceeași secvență de numere aleatorii, dacă generatorul nu este resetat (v. RANDOMIZE).
x < 0 duce la generarea aceleiași secvențe pentru orice x.
x > 0 sau lipsa lui x va duce la generarea următorului număr din secvență.
x = 0 repetă ultimul număr aleatoriu generat.

SGN(x)

Întoarce semnul argumentului x astfel:
dacă x > 0 SGN(x) = 1;
dacă x = 0 SGN(x) = 0;
dacă x < 0 SGN(x) = —1.

SIN(x)

Are ca rezultat sin(x) în simplă precizie (x în radiani)

SPACE$(x).

Întoarce un șir de spații de lungime x. (x este rotunjit și trebuie să fie cuprins în domeniul 0—255).

SPC(i)

Se afișează „i" spații la terminal. SPC poate fi folosit numai în instrucțiuni PRINT și LPRINT.

SQR(x)

Întoarce rădăcina patrată din x (x > = 0).

STR$(x)

Produce un șir de cifre care reprezintă valoarea x.

STRING$(i, j)
STRING$(i, x$)

Produce un șir de lungime i, în care toate caracterele au codul ASCII j sau codul primului caracter din șirul x$.

TAB(i)

Introduce spații pe linia afișată pînă în poziția i. Dacă poziția curentă de afișare a depășit poziția i, se introduc spații pînă la poziția i de pe linia următoare.

TAN(x)

Produce valoarea tangentei lui x. Argumentul x trebuie dat în radiani, iar rezultatul funcţiei este în simplă precizie.

USR[cifra] (x)

Apelează o funcţie scrisă de utilizator în limbaj maşină şi îi transmite un argument „x". ⟨cifra⟩ aparţine intervalului 0—9 şi este aceeaşi cu cea din instrucţiunea DEFUSR corespunzătoare subrutinei. Implicit ⟨cifra⟩ = 0.

VAL(x$)

Produce valoarea numerică a şirului x$. VAL ignoră spaţiile, ⟨LF⟩ şi ⟨CR⟩ din şirul argument.

VARPTR (var)
VARPTR (#nfişier)

Prima formă întoarce adresa primului octet al variabilei ⟨var⟩, care trebuie să aibă atribuită o valoare. Adresa este cuprinsă între —32768 şi 32767; dacă se obţine o adresă negativă, se va aduna cu 65536 pentru a obţine adresa reală. Este folosită pentru a transmite adresa unei variabile BASIC la o subrutină în limbaj de asamblare. Forma a doua dă adresa de început a zonei buffer de I/E alocate pentru fişierul secvenţial sau în acces direct identificat prin numărul ⟨nfişier⟩.

11.3.2. Utilizarea de funcţii şi subrutine în limbaj maşină

Dacă este necesar, se pot scrie secvenţe în limbaj maşină, apelate din interpretorul MBASIC fie ca funcţii cu un singur argument (cu numele USR0,..USR9), fie ca subrutine cu oricîte argumente (prin instrucţiunea CALL).

Utilizarea de subrutine sau funcţii în limbaj maşină din BASIC necesită următoarele:

— cunoaşterea convenţiilor de transmitere a argumentelor (şi rezultatului, în cazul funcţiilor) folosite de interpretor;

— alocarea de memorie pentru secvenţele scrise în limbaj maşină, la sfîrşitul zonei TPA, prin folosirea opţiunii /M la apelarea interpretorului;

— introducerea secvenţelor în cod maşină în locaţiile de memorie alocate, de exemplu prin instrucţiuni POKE;

— apelarea funcţiilor sau subrutinelor încărcate în memorie prin USRn sau CALL.

Pentru alocarea de memorie trebuie cunoscute adresa de sfîrşit a zonei TPA pentru sistemul folosit şi lungimea secvenţelor în limbaj maşină; adresa indicată la opţiunea /M este egală cu diferenţa lor. Se poate aloca memorie în mod acoperitor, dar în felul acesta se limitează lungimea maximă a programelor BASIC, care este determinată de memoria disponibilă pentru interpretor.

Încărcarea în memorie a subrutinelor scurte în limbaj mașină se poate face chiar din programul BASIC, prin instrucțiuni POKE, înainte de apelarea subrutinei sau funcției (se introduc codurile interne de instrucțiuni în zecimal sau în hexazecimal).

Secvențele de cod mai lungi se pot încărca în memorie, înainte de încărcarea interpretorului BASIC, la sfîrșitul zonei TPA, prin mai multe metode:

— folosind un depanator (DDT, SID, ZSID) și comanda A pentru asamblare și încărcare cod simbolic la adresele alocate;

— folosind asamblorul M80 și linkeditorul L80, care încarcă programul la adresele alocate;

— folosind asamblorul ASM (MAC), cu încărcare la adrese mici dintr-un fișier COM, urmată de mutarea la adrese mari printr-o secvență inițială care precede subrutina propriu-zisă.

Interpretorul BASIC se încarcă la începutul zonei TPA (la 100H), fără să afecteze locațiile de la sfîrșitul zonei TPA.

Convențiile de transmitere a argumentelor sînt diferite pentru funcții și pentru subrutine, dar în cazul unui singur argument întreg ele coincid.

În cazul subrutinelor apelate prin CALL se folosește convenția Fortran (Microsoft) de transmitere a parametrilor:

— Dacă numărul parametrilor este mai mic sau egal cu 3, adresele lor sînt transferate prin registrele HL, DE, BC.

— Dacă numărul parametrilor este mai mare ca 3, transferul se face după cum urmează:

— adresa parametru 1 în HL;

— adresa parametru 2 în DE;

— adresele parametrilor 3, 4, 5,... într-un bloc continuu de memorie indicat de registrele BC (BC indică adresa octetului cel mai puțin semnificativ din acest bloc).

În cazul funcțiilor de tip USRn, pentru unicul argument al funcției și pentru rezultatul funcției se folosesc următoarele convenții:

— Registrul A conține o valoare ce specifică tipul argumentului:

2 — un întreg pe doi octeți (în complement față de 2);

3 — un șir;

4 — un număr simplă precizie în virgulă mobilă;

8 — un număr dublă precizie, virgulă mobilă.

— Pentru argumente și rezultate numerice registrele HL conțin adresa zonei unde se află argumentul și rezultatul (numită FAC).

— Pentru numere întregi FAC—3 conține octetul inferior și FAC—2 conține octetul superior al numărului.

— Pentru numere în simplă precizie virgulă mobilă, FAC—3 conține octetul inferior al mantisei, FAC—2 octetul următor al mantisei, iar FAC—1 cei mai semnificativi 8 biți. Bitul 7 al celui mai semnificativ octet este bit de semn (0 = pozitiv, 1 = negativ). FAC conține expo-

nentul — 128. Punctul zecimal este la stînga celui mai semnificativ bit al mantisei.

— Pentru numere în virgulă mobilă dublă precizie FAC—7 pînă la FAC—4 conțin încă 4 octeți ai mantisei.

— Pentru șiruri registrele DE conțin adresa a trei octeți numiți „descriptor de șir". Octetul 0 conține lungimea șirului (0—255), iar octeții 1 și 2 conțin adresa de început a șirului.

Dacă argumentul este un șir literal din program, descriptorul poate da referință către textul programului. Pentru a evita apariția unor rezultate imprevizibile se adaugă un spațiu la sfîrșitul șirului din program.

11.3.3. Comenzi grafice în BASIC-80

Comenzile următoare sînt implementate in cîteva versiuni ale interpretorului BASIC Microsoft pentru Felix M—118 și TPD.

VIEWPORT x1, x2, y1, y2

Stabilește coordonatele zonei imagine (numită și fereastră ecran). Fereastra ecran este un dreptunghi cu colțul din stînga jos în punctul de coordonate (x1, y1) și cu colțul din dreapta sus în punctul (x2, y2). Coordonatele x1, x2, y1, y2 sînt coordonate ecran exprimate în număr de puncte:

$0 <= x1 < x2 < 142$

$0 <= y1 < y2 < 100$

Pe parcursul unui program se poate redefini fereastra ecran, dar la un moment dat există o singură fereastră ecran, stabilită prin ultima comandă VIEWPORT.

WINDOW x1, x2, y1, y2

Stabilește coordonatele ferestrei virtuale (numită și fereastră de date). Fereastra virtuală definește domeniul datelor folosite în reprezentările grafice; vor fi desenate pe ecran numai punctele cu abscisă între x1 și x2 și ordonată între y1 și y2. Coordonatele ferestrei de date pot avea orice valori întregi sau reale.

Punctele din fereastra virtuală se proiectează pe ecran în zona imagine, definită cu VIEWPORT printr-o scalare și o translație. Descrierea imaginilor desenate se face în coordonate virtuale din cadrul ferestrei virtuale.

MOVE x, y

Mută spotul în punctul de coordonate virtuale absolute (x,y).

RMOVE dx, dy

Mută spotul în punctul cu coordonate virtuale relative (dx, dy) față de punctul curent (punctul de plecare).

DRAW x, y

Desenează o dreaptă din punctul curent pînă în punctul cu coordonate virtuale absolute (x, y).

RDRAW dx, dy

Desenează o dreaptă din punctul curent pînă în punctul cu coordonate virtuale relative (dx, dy) față de punctul curent.

ROTATE u

Rotație cu unghiul „u" a liniilor desenate în coordonate relative. Unghiul „u" este exprimat în radiani; valori pozitive indică o rotație în sens trigonometric direct (invers acelor de ceas). Efectul comenzii ROTATE rămîne valabil pînă la următoarea comandă ROTATE. Readucerea în starea inițială se face cu ROTATE 0.

Pentru desenare este necesar ca ecranul să fie în mod „pagină". Aducerea în mod pagină și ștergerea ecranului se fac cu secvențe de control interpretate de sistem:

ștergere: PRINT CHR$(27) + "I"
schimbă mod: PRINT CHR$(27) + "2"

11.4. Exemple de programe BASIC

1) Subrutină de afișare a maximului dintre trei numere

```
10 INPUT a,b,c
20 IF a > b THEN IF a > c THEN PRINT a ELSE PRINT c
   ELSE IF b > c THEN PRINT b ELSE PRINT c
30 RETURN
```

sau:

```
10 INPUT a,b,c
20 IF a > b THEN 30 ELSE 50
30 IF a > c THEN PRINT a ELSE 70
40 GOTO 80
50 IF b > c THEN PRINT b ELSE 70
60 GOTO 80
70 PRINT c
80 RETURN
```

2) Utilizarea de funcții predefinite: subrutină de citire a unui caracter și afișare în ecou, cu transformarea caracterelor de control într-o secvență de două caractere afișabile (↑ x)

```
700 C$=INPUT$(1) : T%=ASC(C$) : RETURN
710 GOSUB 700
720 IF T%>31 THEN PRINT C$  ELSE PRINT "↑"+CHR$(T%+&H40)
```

3) Utilizare funcții pe șiruri de caractere: citirea și verificarea unui specificator de fișier CP/M, cu adăugarea extensiei implicite BAS, dacă lipsește extensia explicită

```
10 INPUT "Nume fisier:",F$
20 IF F$="$" THEN STOP
30 IF LEN(F$) > 10 THEN 80
40 IF MID$(F$,2,1)=":" THEN DISC$=LEFT$(F$,1) ELSE 60
50 IF DISC$ <>"A" AND DISC$ <> "B" GOTO 80
60 IF INSTR(F$,".")=0 THEN F$=F$+".BAS"
70 PRINT F$: GOTO 10
80 PRINT "Nume eronat": GOTO 10
```

4) Utilizare de funcții predefinite: desenarea graficului unei funcții folosind un caracter

```
10 PI=3.14159
20 FOR X=0 TO 4*PI STEP PI/10
30 PRINT SPC(30+30*SIN(X));"*"
40 NEXT
50 END
```

5) Utilizarea de funcții predefinite: program pentru generarea unei secvențe de numere aleatorii, clasificarea lor pe un număr de intervale și desenarea unei histograme cu numărul de valori din fiecare interval

```
10 DATA 30,20,80
20 DEFINT I,J,K,N
30 DIM K(20)
40 READ NR,NI,KMAX
45 ' generare si clasificare numere
50 D=1/NI
60 RANDOMIZE
70 FOR I=1 TO NR
80 FOR J=1 TO NI
90 IF RND < D*J THEN K(J)=K(J)+1
100 NEXT J
110 NEXT I
130 'desenare histograma
140 FOR J=1 TO NI
150 NA=K(J)*80/KMAX        'numar de asteriscuri
160 FOR I=1 TO NA
170 PRINT "#";
180 NEXT I
190 PRINT
200 NEXT J
210 END
```

6) Program de ordonare a unei liste de numere prin metoda inversării elementelor vecine

```
10 DEFINT N,I
20 DIM T(20)
30 GOSUB 200
40 INV=1
50 WHILE INV
60 INV=0
70 FOR I=2 TO N
80 IF T(I-1) < T(I) THEN 100
90 SWAP T(I-1),T(I): INV=1
100 NEXT
110 WEND
120 FOR I=1 TO N : PRINT T(I): NEXT
130 END
200 'subr de citire lista de numere
210 INPUT "nr de valori:",N
220 FOR I=1 TO N
230 INPUT T(I)
240 NEXT
250 RETURN
```

7) Definirea și utilizarea unei funcții de poziționare cursor pe ecran

```
1000 DEF FNP$(X%,Y%)=CHR$(27)+"1"+CHR$(X%+32) + CHR$ (Y%+32)
2000 PRINT FNP$(2,12) "MAIN FUNCTIONS"
2020 PRINT FNP$(30,12) "DISK OPERATIONS"
2030 PRINT FNP$(58,12) "OTHER"
2040 PRINT FNP$(2,14)" B = BUILD NEW SHEET"
2050 PRINT FNP$(30,14) "R = READ DISK FILE"
2060 PRINT FNP$(58,14) "L = LOAD DISK FILE"
```

8) Crearea unui fișier secvențial

```
10 OPEN "o",#1,"DATE"
20 INPUT "nume";n$
30 IF n$=" " THEN END
40 INPUT "adresa";a$
50 INPUT "data nasterii";b$
60 PRINT#1,n$;",";a$;",";b$
70 PRINT:GOTO 20
```

9) Exploatarea unui fișier secvențial

```
10 OPEN "i",#1,"DATE"
15 IF EOF(1)=-1 THEN END
20 INPUT#1,n$,a$,b$
30 PRINT n$, a$, b$
40 GOTO 20
```

10) Actualizare fişier secvenţial

```
10 ON ERROR GOTO 2000
20 OPEN "i",#1,"DATE"
30 ' daca fisierul exista, se citeste pentru a fi copiat
40 OPEN "o",#2,"copie"
50 IF EOF(1)=-1 THEN 90
60 LINE INPUT#,a$
70 PRINT#2,a$
80 GOTO 50
90 CLOSE #1
100 KILL "date"
110 REM se adauga noi date la fisier
120 INPUT "nume";n$
130 IF n$=" " THEN 200
140 LINE INPUT "adresa";a$
150 LINE INPUT "data nasterii";b$
160 PRINT#2,n$
170 PRINT#2,a$
180 PRINT#2,b$
190 PRINT:GOTO 120
200 CLOSE
205 ' schimba numele fisierului
210 NAME "copie" AS "date"
2000 IF ERR=53 AND ERL=20 THEN OPEN "o",#2,"copie" : RESUME 120
2010 ON ERROR GOTO 0
```

11) Salvarea unei matrice de întregi într-un fişier disc şi recitirea din fişier cu afişare la consolă

```
10 DIM ARR%(10,10)
20 FILE$="TEST.DAT"
30 NL%=5 : NC%=5
50 OPEN "O",#1,FILE$
60 FOR I%=1 TO NL%
70 FOR J%=1 TO NC%
80 WRITE #1,10*(I%-1)+J%
90 NEXT J%: NEXT I%: CLOSE #1
100 OPEN "I",#1,FILE$
110 FOR I%=1 TO NL%: FOR J%=1 TO NC%
120 INPUT #1,ARR%(I%,J%): PRINT ARR%(I%,J%)
130 NEXT J%,I%: CLOSE #1
```

12) Crearea unui fişier de date în acces direct

```
10 OPEN "r",#1,"TEST",42
20 FIELD #1,20 AS n$,14 AS a$,8 AS b$
30 INPUT "nr.inregistrare";cod%
35 IF cod%=0 THEN END
40 INPUT "nume";x$
50 INPUT "adresa";y$
60 INPUT "data nasterii";z$
70 LSET n$ = x$
```

```
80 LSET a$ = y$
90 LSET b$ = z$
100 PUT #1,cod%
110 GOTO 30
```

13) Citirea unui fişier în acces direct

```
10 OPEN "r",#1,"TEST",42
20 FIELD #1,20 AS n$,14 AS a$,8 AS b$
30 INPUT "nr.inregistrare";cod%
35 IF cod%=0 THEN END
40 GET#1,cod%
50 PRINT n$; a$; b$
60 GOTO 30
```

14) Exemplu de program grafic în BASIC

```
01    REM Rotirea unui patrat in jurul unui colt
10    PRINT CHR$(27)+"I" : PRINT CHR$(27)+"2"
20    WINDOW 0,100,0,100
30    VIEWPORT 20,104,0,100
40    FOR K=1 TO 20
50    INPUT N 'numar de rotatii
60    PRINT CHR$(24) 'sterge ecran
70    MOVE 50,50
80    GOSUB 200 'desenare patrat
90    FOR I=1 TO N
100   ROTATE 6.28/N*I
110   GOSUB 200
120   NEXT
130   NEXT
150   REM subrutina desenare patrat
200   RDRAW 0,40
210   RDRAW 40,0
220   RDRAW 0,-40
230   RDRAW -40,0
250   RETURN
```

15) Utilizare subrutină în limbaj maşină (subrutina roteşte la stînga de 4 ori octetul inferior din argument; 7 este codul instrucţiunii RLC)

```
10 SUB=&H9000
20 POKE SUB,&H7E
30 FOR I=1 TO 4: POKE SUB+I,7:NEXT
40 POKE SUB+5,&H77
50 POKE SUB+6,&HC9
60 INPUT A%
70 CALL SUB(A%)
80 PRINT A%
90 GOTO 60
```

10) Utilizare funcție în limbaj mașină (funcția transformă octetul inferior al argumentului, rotit de 4 ori)

```
10 SUB=&H9000
15 DEF USR=SUB
20 POKE SUB,&H7E
30 FOR I=1 TO 4: POKE SUB+I,7:NEXT
40 POKE SUB+5,&H77
50 POKE SUB+6,&HC9
60 INPUT A%
70 PRINT USR(A%)
80 PRINT A%
90 GOTO 60
```

12. UTILIZAREA PROGRAMULUI DE GESTIUNE DBASE

Programul DBASE poate fi considerat fie ca un program destinat aplicaţiilor de gestiune cu volum redus de date, fie ca un sistem simplu de gestiune a bazelor de date relaţionale, fie ca un interpretor pentru un limbaj specializat, de nivel foarte înalt.

Iată cîteva tipuri de aplicaţii, pentru care poate fi utilizat cu succes sistemul DBASE: introducerea şi validarea de date primare, memorarea, regăsirea şi listarea de date din fişiere, ordonarea, prelucrarea complexă (inclusiv calcule) şi întocmirea de rapoarte cu date din fişiere, cu posibilitatea de actualizare a fişierelor etc.

Avantajele principale ale programului DBASE sînt: uşurinţa cu care se învaţă şi se utilizează, puterea comenzilor şi modul de lucru interpretativ, care împreună permit realizarea de noi aplicaţii într-un timp mult mai scurt decît în oricare alt limbaj de programare.

Putem deosebi două moduri diferite de utilizare a programului DBASE, dar care folosesc aceleaşi comenzi:

— Modul imediat, mai uşor accesibil pentru utilizatori, în care o comandă introdusă de la consolă este imediat executată (interpretată) de către sistem; în acest mod utilizatorul este oarecum limitat la posibilităţile şi formele de prezentare prevăzute de comenzile DBASE.

— Modul programat, mai greu accesibil utilizatorilor neprogramatori, în care comenzile referitoare la fişiere sînt combinate cu alte comenzi de intrare—ieşire şi de control în programe structurate, memorate în fişiere de comenzi şi lansate ulterior în execuţie printr-o singură comandă operator; modul programat oferă mai multe posibilităţi în realizarea aplicaţiilor şi a formei de dialog cu operatorul, dar necesită un efort mai mare de concepere şi punere la punct a programelor.

Cele două moduri de utilizare DBASE nu se exclud, deoarece sînt anumite operaţii realizate mai rapid şi mai convenabil în modul imediat: crearea iniţială, modificarea structurii, cereri neanticipate la date din fişiere, inspectarea şi verificarea vizuală a fişierelor, corecţii minore în date ş.a., după cum alte operaţii se pretează mai bine la realizarea prin programe: crearea de rapoarte însoţite eventual de prelucrări,

actualizarea masivă a fişierelor, introducere de date cu validare, extragere de date din mai multe fişiere ş.a.

Comenzile DBASE cu efect asupra fişierelor realizează operaţii care în alte limbaje sînt implementate prin programe separate: listarea selectivă cu format fix sau definit de utilizator a datelor din unul sau două fişiere, editarea prin modificarea selectivă a datelor din fişier, adăugări la sfîrşit, inserări, ştergeri de articole, sortarea în orice ordine a datelor dintr-un fişier, calcule uzuale cu date selectate din tot fişierul, căutarea după conţinut a unor articole, combinarea datelor din fişiere diferite ş.a.

Dezavantajul programului DBASE constă în timpul destul de mare de execuţie a unor programe, datorat interpretării comenzilor şi încărcării repetate a segmentelor DBASE.

Pe de altă parte, în DBASE mai mult decît în alte limbaje de programare, diferenţa dintre timpul de rulare a unor programe echivalente ca efect, dar diferite în conţinut poate fi foarte mare şi deci este cu atît mai importantă stăpînirea unor tehnici de programare care să conducă la optimizarea performanţelor programelor DBASE.

12.1. Facilităţi şi convenţii ale sistemului DBASE

12.1.1. Fişiere şi variabile DBASE

Programul DBASE operează cu mai multe tipuri de fişiere, dintre care cel mai important este tipul „fişier bază de date".

Un *fişier de date* DBASE (de tipul DBF) conţine la început o înregistrare de structură, urmată de un număr oarecare (eventual zero) de înregistrări de date; înregistrările de date au toate acelaşi format şi aceeaşi lungime.

O înregistrare de date este structurată în mai multe cîmpuri de date, fiecare cîmp avînd un nume, o lungime şi un tip de date.

În DBASE există trei tipuri de date:

— şiruri de caractere (C);

— numere întregi sau neîntregi (N);

— logice (L).

Înregistrarea de structură conţine cîteva informaţii generale despre fişier — data ultimei actualizări, număr total de articole, lungimea articolelor — urmate de informaţii referitoare la fiecare cîmp din articol: numele cîmpului, tipul datelor, lungimea totală şi numărul de cifre de la partea fracţionară.

Includerea descrierii structurii în fişierul de date elimină necesitatea descrierii fişierului de către utilizator în programele DBASE.

Majoritatea comenzilor DBASE operează asupra unui singur fişier de date, numit fişier curent (activ) şi care nu este specificat explicit în aceste comenzi.

De asemenea există şi noţiunea de articol curent în cadrul fişierului, fiind prevăzute mai multe comenzi de localizare articol sau de poziţionare pe un articol.

Cîmpurile articolului curent pot fi utilizate ca variabile în diferite comenzi, funcţii şi expresii DBASE.

Uneori sînt însă necesare şi *variabile* care nu reprezintă nume de cîmpuri din fişier, deci variabile independente de fişierul curent şi păstrate în memorie; de exemplu pentru rezultate intermediare şi finale în anumite expresii de calcul.

Numele de cîmpuri şi numele de variabile pot avea maxim 10 caractere: litere, cifre şi caracterul ':'; un nume trebuie să înceapă cu o literă şi să nu se termine cu ':'.

Variabilele de memorie pot avea şi ele unul dintre cele trei tipuri de date (caracter, numeric, logic) şi sînt limitate ca număr de 64. Este posibilă salvarea şi, respectiv, citirea variabilelor de memorie în (din) fişiere speciale, de tipul MEM.

Unui fişier de date DBASE i se pot asocia unul sau mai multe *fişiere index*, de tipul NDX, cîte un fişier index pentru fiecare cîmp indexat (de fapt pentru fiecare cheie de indexare). Un fişier index conţine, pentru fiecare articol al fişierului de date, valoarea din cîmpul indexat şi numărul articolului, în ordinea crescătoare a valorilor din acest cîmp.

Alte tipuri predefinite de fişiere DBASE sînt prevăzute pentru memorarea unor comenzi de formatare a ecranului sau imprimantei (tip FMT), pentru memorarea unor formulare de raportare (tip FRM) şi pentru memorarea rezultatelor comenzilor DBASE afişate în mod normal pe ecran (tip TXT).

O categorie aparte de fişiere DBASE sînt *fişierele de comenzi* (de tip CMD), care conţin programe formate din comenzi DBASE; aceste fişiere pot fi create sub DBASE printr-o comandă specială sau separat de DBASE, cu orice editor de texte.

Fişierele de comenzi pot avea orice lungime, dar fişierele de date DBASE au următoarele limitări:

maximum 65535 de articole în fişier;

maximum 32 de cîmpuri într-un articol;

maximum 1000 de caractere într-un articol;

maximum 254 de caractere într-un cîmp, într-o variabilă sau într-o constantă de tip şir (tip C);

maximum 10 cifre semnificative pentru date numerice (cîmpuri variabile, constante);

între 10**—63 şi 1.8*10**63 domeniul de mărime al numerelor.

La un moment dat pot fi deschise simultan 16 fişiere de diferite tipuri, dar numai două fişiere de date DBF.

12.1.2. Expresii şi funcţii DBASE

Expresiile DBASE se utilizează mai ales în anumite comenzi conţinute în programe.

O expresie poate fi formată din nume de cîmpuri, variabile de memorie, constante, funcţii şi operatori. Toate componentele unei expresii trebuie să fie de acelaşi tip, care este şi tipul expresiei (C, N, L).

Constantele de tip caracter sînt şiruri de maximum 254 de caractere încadrate între ghilimele simple, ghilimele duble sau paranteze drepte; sînt prevăzute trei categorii de caractere delimitator, pentru a se alege acelea care nu apar în şirul delimitat.

Operatorii care pot fi aplicaţi asupra datelor de tipul C sînt operatori de concatenare şiruri şi operatori de relaţie; de asemenea, se pot aplica funcţii care au ca argumente şiruri. Ordinea de evaluare în expresiile de tipul caracter: paranteze şi funcţii, relaţii, $, + şi — (concatenare).

Operatorii de concatenare sînt:

+ alăturare de şiruri;

— alăturare de şiruri cu mutarea blancurilor finale din primul şir la sfîrşitul şirului rezultat (blancurile din faţă sau conţinute în şir rămîn neschimbate).

Operatorii de relaţie sînt:

<,>,=,#,<=,>=, $

Operatorul '$' este diferit de funcţia '$' şi are următoarea interpretare: A$B este adevărat, dacă şirul A este conţinut în şirul B sau este egal cu B.

Constantele de tipul numeric sînt numere zecimale întregi sau neîntregi, cu semn, avînd o precizie maximă de 10 cifre.

Operatorii care pot fi aplicaţi asupra datelor numerice sînt operatorii aritmetici, de relaţie precum şi funcţiile cu argument numeric.

Operatorii aritmetici din DBASE sînt:

+, —, *, /, ()

Ordinea de evaluare în expresiile numerice şi de relaţie este: paranteze şi funcţii, semnul plus şi minus, * şi / ,+ şi —, relaţii.

Rezultatul unei expresii numerice are la partea fracţionară atîtea cifre cît are operandul cu număr maxim de cifre după punctul zecimal.

Constantele logice pot fi reprezentate prin mai multe caractere astfel:

adevărat: „T", „t", „Y", „y"
fals: „F", „f", „N", „n"

Expresiile de relaţie au un rezultat logic, după cum relaţia respectivă este adevărată sau falsă.

Operatorii logici pot avea ca operanzi expresii cu rezultat logic, deci constante şi variabile logice sau expresii de relaţie.

Operatorii logici se notează în DBASE ca și în Fortran:

.NOT., .AND., .OR. (în ordinea descrescătoare a priorității)

Funcțiile DBASE pot fi clasificate după tipul rezultatului (C,N,L) sau după rolul îndeplinit. Urmează descrierea sumară a funcțiilor existente în DBASE (nu se pot defini alte funcții).

Lista funcțiilor DBASE

INT (expn)

are ca rezultat partea întreagă a expresiei numerice ⟨expn⟩, prin trunchierea părții fracționare.

VAL (șir)

are ca rezultat valoarea numerică a șirului, șir care trebuie să conțină doar cifre, semn și punct zecimal (este o conversie de la tipul șir la tipul numeric).

CHR (expn)

are ca rezultat caracterul al cărui cod ASCII este egal cu valoarea expresiei numerice ⟨expn⟩; se folosește pentru caractere de control, neafișabile.

STR (expn,lung,[zec])

are ca rezultat un șir de caractere de lungime ⟨lung⟩ cu ⟨zec⟩ cifre zecimale la partea fracționară, șir echivalent cu valoarea expresiei numerice ⟨expn⟩ (este o conversie de la tipul numeric la tipul caracter). Ultimii doi parametri pot fi constante, variabile sau expresii.

$(expc,start,lung)

are ca rezultat un subșir extras din șirul definit prin expresia de tipul caracter ⟨expc⟩, începînd din poziția ⟨start⟩ și cu ⟨lung⟩ caractere. Ultimii doi parametri pot fi constante, variabile sau expresii.

@ (șir1,șir2)

are ca rezultat un număr întreg ce reprezintă poziția caracterului din ⟨șir2⟩, unde începe un subșir identic cu ⟨șir1⟩, sau zero dacă ⟨șir2⟩ nu conține ⟨șir1⟩.

! (expc)

are ca rezultat un șir care provine din șirul definit prin expresia de tip caracter ⟨expc⟩, în care toate literele mici au fost înlocuite prin literele mari echivalente.

LEN (șir)

are ca rezultat un întreg egal cu lungimea șirului ⟨sir⟩.

TRIM (șir)

are ca rezultat un șir care provine din șirul ⟨sir⟩ prin eliminarea blancurilor finale.

#

are ca rezultat numărul articolului curent din fișierul de date curent.

*

are ca rezultat valoarea adevărat („Y", „T"), dacă articolul curent este marcat pentru ștergere prin comanda DELETE, și valoarea fals, dacă articolul nu a făcut obiectul unei comenzi DELETE.

EOF

are un rezultat logic cu valoarea adevărat, dacă s-a ajuns la sfîrșitul fișierului curent, și cu valoarea fals, în caz contrar.

TYPE (exp)

are ca rezultat un caracter din următoarele 4: „C", „N", „L", „U", caracter care reprezintă tipul expresiei ⟨exp⟩; caracterul „U" (undefined) rezultă pentru cîmpuri inexistente în fișierul curent.

FILE (exp)

are un rezultat logic cu valoarea adevărat, dacă există fișierul specificat prin expresia de tip caracter ⟨expc⟩, și cu valoarea fals, în caz contrar.

DATE()

are ca rezultat un șir de 8 caractere, cu formatul ZZ/LL/AA, reprezentînd data calendaristică introdusă și memorată de DBASE (ZZ = zi, LL = luna, AA = an).

12.1.3. Convenții ale limbajului de comenzi DBASE

O comandă DBASE este un șir de maxim 254 de caractere, care începe obligatoriu cu numele comenzii (un verb sau un caracter special) și este eventual urmat de mai mulți parametri ai comenzii, separați între ei prin oricîte spații (blancuri).

O serie de parametri sînt cuvinte cheie (impuse) ale limbajului DBASE, iar alți parametri trebuie precedați de anumite cuvinte cheie, care indică semnificația parametrului și care apropie comenzile DBASE de propoziții imperative ale limbii engleze.

Parametrii cuvinte cheie sau precedați de cuvinte cheie pot apărea în general în orice ordine în cadrul unei comenzi. Exemplu:

```
SORT ON COD TO PERSORD ASCENDING
SORT ASCENDING TO PERSORD ON COD
```

Atît numele de comenzi cît și cuvintele cheie pot fi prescurtate la primele 4 litere sau mai multe și pot fi scrise cu litere mari sau cu litere mici, deoarece DBASE nu face diferență între litere mari și litere mici. Exemplu:

```
display structure
DISPLAY STRUCTURE
Displ Struct
DISP STRU
```

Deşi este posibilă folosirea cuvintelor cheie şi verbelor ca nume de fişiere, de cîmpuri sau de variabile, nu se recomandă această practică.

Fiecare comandă ocupă o linie şi nu se pot scrie mai multe comenzi într-o linie. Exemplu de comandă incorectă:

```
IF EOF STORE 'T' TO END
ENDIF
```

Secvenţa de comenzi corectă este:

```
IF EOF
STORE 'T' TO END
ENDIF
```

În comenzile care cer un anumit tip de fişiere DBASE se poate scrie doar numele fişierului, extensia numelui fiind implicită. Exemple:

USE PRODUSE în loc de USE PRODUSE.DBF

INDEX ON COD TO XCOD în loc de INDEX ON COD TO XCOD.NDX

DO LISTARE în loc de DO LISTARE.CMD

În principiu orice comandă DBASE se poate folosi atît direct de la consolă, cît şi în fişiere de comenzi, dar există anumite comenzi folosite aproape exclusiv în dialogul direct utilizator—sistem (HELP, CREATE, EDIT, CHANGE, BROWSE, INSERT, ş.a.) şi alte comenzi întîlnite numai în programe (@, ACCEPT, INPUT, DO, IF, LOOP, RETURN, NOTE ş.a.).

Majoritatea comenzilor de prelucrare şi de afişare a datelor dintr-un fişier au un *domeniu de acţiune* al comenzii, specificat explicit în comandă sau cu valoare implicită (dar diferite comenzi pot avea diferite domenii implicite de acţiune).

Domeniul de acţiune al unei comenzi poate fi specificat prin următoarele cuvinte cheie:

ALL toate articolele din fişier (indiferent de poziţia curentă);

NEXT n următoarele ⟨n⟩ articole din fişier, inclusiv articolul curent;

RECORD n numai articolul cu numărul ⟨n⟩;

FOR exp toate articolele pentru care este adevărată expresia ⟨exp⟩;

WHILE exp toată secvenţa de articole care urmează articolului curent şi satisfac expresia ⟨exp⟩, pînă la primul articol care nu satisface condiţia ⟨exp⟩.

În cursul introducerii comenzilor DBASE se pot folosi o serie de *caractere de control* ale sistemului CP/M:

RUB, ^H(BS) şterge ultimul caracter introdus

^S, ^Q oprire şi reluare afişare ca rezultat al unei comenzi

^P ecou la imprimantă de la consolă

^X, ^U şterge linia curentă

De asemenea, se poate folosi caracterul ESC (Escape) pentru terminarea forțată a execuției unor comenzi și revenirea în DBASE.

O serie de comenzi DBASE sînt concepute să opereze în mod ecran, dar ele pot fi folosite și în mod defilare, dacă nu s-a făcut instalarea programului DBASE pentru terminalul folosit.

Alegerea modului de lucru, precum și stabilirea altor opțiuni și parametri de lucru ai programului DBASE se pot face printr-o comandă specială (comanda SET), care se poate folosi de mai multe ori într-o sesiune de lucru sau într-un program DBASE.

O facilitate a limbajului DBASE, deosebit de utilă în parametrizarea sau prescurtarea comenzilor, o constituie *macrosubstituția* unor variabile de tip caracter.

O variabilă de tip C, precedată de semnul ampersand '&', este înlocuită prin valoarea sa, care poate fi un șir de caractere de orice lungime cu orice semnificație: o constantă, un nume de cîmp, un nume de fișier sau numele altei variabile. Este permisă chiar o substituție repetată folosind mai multe caractere '&'.

Toate substituțiile se efectuează înainte de interpretarea comenzii în care apar. Exemplu:

```
STORE '7' TO X
STORE 'X' TO Y
STORE 'Y' TO Z
```

Variabila X are valoarea 7, iar variabila Y are valoarea 'X'.

```
? X și ? &X      afișează 7
? &Y  afișează 7
? &Z  afișează X
? &&Z afișează 7
```

Folosind variabile care urmează a fi substituite, este posibil ca unele comenzi să aibă un caracter mai general, putînd opera cu orice fișier sau cîmp sau variabilă. Exemple:

a) ACCEPT 'Nume fișier' TO Fișier
   USE &Fișier

dacă variabila ⟨Fișier⟩ primește valoarea 'PRODUSE', atunci comanda anterioară este echivalentă cu comanda:

```
USE PRODUSE
```

b) STORE CODCL + CODPR TO Cheie
   FIND &Cheie

valoarea variabilei ⟨Cheie⟩ rezultă aici din concatenarea șirurilor ce reprezintă valorile din cîmpurile CODCL și CODPR.

Așa cum se vede și din exemplele anterioare, șirul care substituie variabila poate fi citit de la consolă sau poate rezulta dintr-o expresie calculată prin program.

Este posibil ca șirul ce substituie o variabilă să fie concatenat la sfîrșit cu un alt șir constant (fără ghilimele) dacă se pune caracterul

punct '.' după numele variabilei (punctul dispare după concatenare). Exemplu:

USE & Disc. : CLIENTI

dacă variabila ⟨Disc⟩ primeşte valoarea 'B', atunci comanda anterioară devine:

USE B: CLIENTI

Macrosubstituția se poate folosi pentru orice element al unei comenzi, inclusiv pentru numele comenzii. Exemplu:

STORE „ACCEPT 'pentru continuare apăsați CR' TO NULL"
TO Pauza.

O comandă de forma:

& Pauza

este de fapt o prescurtare a comenzii următoare:

ACCEPT 'pentru continuare apăsați CR' TO NULL

şi este utilă atunci cînd această comandă apare de mai multe ori într-un program.

Uneori valoarea care substituie numele variabilei este folosită ca o constantă de tip şir de caractere şi trebuie încadrată între ghilimele. Exemple:

a) IF .NOT. FILE ('& Fişier')
 ? 'Nu există fişierul & Fişier'
 ENDIF

b) IF '& Fişier1' = '& Fişier2'
 ? 'Eroare'
 ENDIF

12.2. Comenzi DBASE utilizate direct de la consolă

12.2.1. Comenzi de creare, listare, căutare şi utilitare

Înainte de a folosi programul DBASE în realizarea unor aplicații orice utilizator începe prin a exersa direct de la consolă comenzile de bază, pentru a se familiariza cu modul de lucru al programului, pentru a estima posibilitățile sale.

Dacă nu se dispune de un fişier DBF deja creat, atunci trebuie început cu definirea structurii unui fişier de date şi cu introducerea unor date de test în acest fişier (comanda CREATE).

Se vor încerca apoi comenzile de deschidere şi de închidere a unui fişier de date (comanda USE) precum şi comenzile de listare a conținutului fişierului creat (comenzile LIST, DISPLAY).

Adăugarea de noi date la fişier se poate face prin comenzile APPEND şi INSERT, iar ştergerea de articole prin comanda DELETE; se va remarca modul mai deosebit de realizare a ştergerii de articole, în două etape: prima etapă de marcare pentru ştergere şi invali-

dare (comanda DELETE) și a doua etapă de eliminare efectivă a articolelor marcate din fișier (comanda PACK). Între cele două etape este posibilă recuperarea de articole din cele marcate pentru ștergere, prin anularea ștergerii (comanda RECALL).

Comenzile utilitare DBASE permit realizarea unor operații de inventariere fișiere (LIST FILES), de copiere fișiere (COPY), de ștergere fișiere (DELETE FILE), de schimbare a numelui unui fișier (RENAME) și de recitire a tabelului director după schimbarea unui disc (RESET), fără a ieși din DBASE în sistem.

Selectarea de articole pe baza numărului de articol în fișier (comanda GO/GOTO), trecerea de la un articol la altul (comanda SKIP) și localizarea unui articol pe baza conținutului său (comanda LOCATE) permit căutarea și stabilirea unui articol curent.

Editorul de date încorporat în DBASE și activat prin comanda EDIT asigură posibilitatea de vizualizare individuală a datelor dintr-un articol (împreună cu numele cîmpurilor) cunoscînd numărul articolului, precum și posibilitatea de modificare selectivă a datelor afișate: se poate de asemenea „răsfoi" fișierul, mergînd din articol în articol înainte sau înapoi.

Comanda BROWSE oferă o altă formă de vizualizare și de editare a datelor dintr-un fișier, printr-o „fereastră" care proiectează pe ecran numai anumite articole și cîmpuri de articole vecine; această fereastră poate fi deplasată în cadrul fișierului considerat ca un tabel plan în care fiecare linie este un articol și fiecare coloană este un cîmp.

Folosind comanda SET SCREEN OFF/ON se va constata diferența dintre lucrul în mod ecran și lucrul în mod defilare la comenzile APPEND, INSERT, EDIT și CREATE.

Urmează descrierea sumară a comenzilor menționate, împreună cu unele comentarii privind utilizarea lor.

CREATE [fișier]

Creează înregistrarea de structură a unui nou fișier DBF și, eventual, mai multe articole de date pe baza datelor introduse de la consolă. La definirea structurii se introduce pentru fiecare cîmp de articol, în ordinea memorării lor pe disc, numele, tipul, lungimea și, pentru numere neîntregi, numărul de cifre zecimale. Este important ca structura fișierului să fie bine gîndită înaintea introducerii datelor pentru a evita redefinirea structurii, care înseamnă de fapt crearea unui alt fișier. Introducerea de date în fișier se poate face atît la comanda CREATE, cît și prin comenzi APPEND ulterioare, cu același efect. Fișierul creat nu devine automat fișier curent, fiind necesară deschiderea și activarea lui prin USE.

USE [fișier]

Deschide fișierul de date ⟨fișier⟩, care devine fișier curent pentru comenzile următoare. Extensia DBF este implicită. Comanda USE

fără nume de fişier închide fişierul curent; dar trebuie reţinut că şi deschiderea unui nou fişier închide automat fişierul activ curent.

USE fişier INDEX fişier index [,...]

Deschide un fişier de date ⟨fişier⟩ şi unul sau mai multe fişiere index asociate; numai primul fişier ⟨fişier index⟩ este folosit pentru comenzile de căutare, listare şi actualizare.

APPEND

Permite adăugarea de noi articole la sfîrşitul fişierului curent cu date introduse de la consolă; se afişează la fiecare articol numărul articolului şi numele cîmpurilor. Se iese din modul adăugare prin introducerea unui caracter CR la început de articol. Se pot folosi următoarele caractere de control:

^E, ^A	poziţionare pe cîmpul precedent
^X, CR	poziţionare pe cîmpul următor
^S	poziţionare pe caracterul precedent
^D	poziţionare pe caracterul următor
^G	şterge caracterul curent
RUB	şterge caracterul precedent
^Y	şterge cîmpul curent
^C	poziţionare pe articolul următor

APPEND BLANK

Adaugă un singur articol nul, numai cu spaţii, la sfîrşitul fişierului curent.

APPEND FROM fişier [SDF] [FOR exp]
[DELIMITED]

Adaugă la sfîrşitul fişierului curent articolele din fişierul ⟨fişier⟩ care îndeplinesc condiţia ⟨exp⟩; clauza SDF (System Data Format) arată că este un fişier non-DBASE, iar clauza DELIMITED precizează caracterul care separă articolele în fişierul SDF (dacă există un delimitator de articole). Nu este necesar ca cele două fişiere să aibă aceeaşi structură şi nici ca lungimea cîmpurilor comune să fie aceeaşi.

INSERT [BEFORE] [BLANK]

Inserează un nou articol după articolul curent, sau înaintea articolului curent, dacă se foloseşte opţiunea BEFORE. Dacă fişierul curent este indexat şi indexul activ, INSERT este echivalent cu APPEND. Inserarea într-un fişier mare poate dura mult şi trebuie evitată. INSERT BLANK inserează un articol cu blancuri după sau înainte de articolul curent.

DISPLAY [domeniul] [FOR exp] [cîmp,...]
LIST [domeniul] [FOR exp] [cîmp,...]

Listează la consolă articolele din fişierul curent din domeniul specificat care îndeplinesc expresia condiţională ⟨exp⟩; se afişează toate cîmpu-

rile specificate în lista opțională de cîmpuri. Domeniul implicit este tot fișierul pentru LIST și numai articolul curent pentru DISPLAY. La DISPLAY ALL afișarea se oprește după fiecare 15 articole, iar pentru continuare se apasă clapa CR.

DISPLAY STRUCTURE
LIST STRUCTURE

Afișează la consolă conținutul înregistrării de structură a fișierului curent.

DISPLAY MEMORY
LIST MEMORY

Afișează numele și valorile variabilelor de memorie existente în momentul respectiv.

DELETE [domeniul] [FOR exp]

Marchează pentru ștergere articolele din domeniul indicat al fișierului curent, care satisfac condiția ⟨exp⟩. Domeniul implicit este articolul curent. Articolele marcate apar la listarea fișierului precedate de un asterisc, dar sînt ignorate la copiere, adăugare sau sortare.

RECALL [domeniul] [FOR exp]

Recuperează articolele marcate pentru ștergere din domeniul specificat, care satisfac condiția ⟨exp⟩, prin anularea efectului comenzii DELETE asupra acestor articole.

PACK

Compactare fișier curent prin eliminarea articolelor marcate pentru ștergere prin DELETE. Articolele eliminate nu mai pot fi recuperate. Eliminarea se poate face și prin copiere cu COPY. Compactarea fișierelor indexate poate dura mult, fiind preferabilă copierea și reindexarea fișierului.

DISPLAY FILES [ON disc] [LIKE *.tip]
LIST FILES [ON disc] [LIKE *.tip]

Afișează la consolă numele fișierelor de tip DBF de pe discul curent, dacă nu se folosesc alte opțiuni. Opțiunea ON poate preciza discul suport, iar opțiunea LIKE poate selecta un grup de fișiere de orice tip pentru afișare.

DELETE FILE fișier

Șterge fișierul ⟨fișier⟩ de tipul implicit DBF sau de tipul specificat. Fișierul trebuie să fie închis.

RENAME ⟨fișier1⟩ TO ⟨fișier2⟩

Schimbare nume fișier ⟨fișier1⟩ în numele ⟨fișier2⟩. Extensia implicită este DBF. Fișierul nu trebuie să fie deschis.

COPY TO ⟨fişier⟩ [domeniu] [FIELD cîmp, . .] [FOR exp] [SDF] [DELIMITED [WITH caracter]]

Copiază din fişierul curent în fişierul specificat cîmpurile din lista FIELD (implicit toate cîmpurile pentru articolele selectate prin ⟨domeniu⟩) şi prin condiţia ⟨exp⟩ (implicit toate articolele). Opţiunea SDF arată că noul fişier nu este un fişier DBASE (nu conţine o înregistrare de structură), iar opţiunea DELIMITED arată prin ce caracter separator sînt delimitate articolele din fişierul SDF (implicit prin virgulă); şirurile de caractere sînt încadrate automat între ghilimele în fişierul SDF. Comenzile COPY şi APPEND cu opţiunea SDF permit transferul de fişiere între DBASE şi programe scrise în alte limbaje.

COPY TO fişier STRUCTURE

Copiază înregistrarea de structură din fişierul curent în fişierul ⟨fişier⟩ creat prin comanda COPY.

n
GO/GOTO RECORD n
GO/GOTO var

Poziţionare pe articolul cu numărul ⟨n⟩ sau al cărui număr este dat de valoarea variabilei numerice ⟨var⟩, din fişierul curent.

GO/GOTO [TOP] [BOTTOM]

Poziţionare pe început (TOP) sau sfîrşit (BOTTOM) de fişier curent. Imediat după deschidere primul articol este articol curent.

SKIP [+] [—] [exp]

Salt înainte (+) sau înapoi (—) peste numărul de articole specificat prin expresia numerică ⟨exp⟩; dacă lipseşte ⟨exp⟩, se sare la articolul următor sau precedent.

LOCATE [domeniul] [FOR exp]

Localizează articolele din domeniul specificat al fişierului curent care satisfac condiţia ⟨exp⟩. Domeniul implicit este tot fişierul. Efectul comenzii este poziţionarea pe primul articol care satisface condiţia; la celelalte articole care mai satisfac aceeaşi condiţie se poate ajunge prin comanda CONTINUE. Căutarea este secvenţială, înainte, şi de aceea se poate termina cu poziţionare pe sfîrşit de fişier, dacă nu există nici un articol care satisface condiţia ⟨exp⟩. Se foloseşte pentru fişiere neindexate sau cu index inactiv.

CONTINUE

Poziţionare pe următorul articol care satisface condiţia conţinută în ultima comandă LOCATE sau pe sfîrşit de fişier, dacă nu mai există nici un articol care să satisfacă condiţia.

EDIT [n]

Se intră în modul editare, afişîndu-se datele din articolul ⟨n⟩ împreună cu numele cîmpurilor. Dacă se lucrează în mod ecran, atunci se poate

poziţiona cursorul pe orice cîmp şi pe orice caracter şi se pot face modificări prin înlocuire sau inserare. După editarea unui articol se poate trece la articolul următor sau precedent şi se rămîne în modul editare pînă la ieşirea cu CTRL/W sau CTRL/Q. Caractere de control în mod ecran la EDIT:

CR	poziţionare pe cîmpul următor sau pe articolul următor
↑E, ↑A	poziţionare pe cîmpul precedent
↑K, ↑F	poziţionare pe cîmpul următor
↑S	poziţionare pe caracterul precedent
↑D	poziţionare pe caracterul următor
↑G	şterge caracterul din dreptul cursorului
RUB	şterge caracterul din stînga cursorului
↑Y	şterge cîmpul curent
↑C	poziţionare pe articolul următor
↑R	poziţionare pe articolul precedent
↑U	marcare pentru ştergere sau recuperare articol curent
↑W	scrie în fişier articolul curent şi ieşire din EDIT
↑Q	ieşire din EDIT fără modificarea articolului curent

BROWSE

Permite afişarea şi editarea articolelor din fişierul curent care sînt decupate printr-o „fereastră" în fişier. Iniţial fereastra se află în colţul din stînga sus, deci cuprinde primele cîmpuri din primele articole ale fişierului. În fereastră se afişează primele 19 articole (sau mai puţine) şi atîtea cîmpuri cîte încap complet pe o linie de ecran. Se pot utiliza caracterele de control de la comanda EDIT: ↑E, ↑A, ↑X, ↑F, ↑D, ↑S, ↑G, RUB, ↑Q, ↑W, ↑C, ↑R, ↑U. În plus se mai pot folosi următoarele caractere:

↑B deplasare fereastră la stînga cu un cîmp
↑Z deplasare fereastră la dreapta cu un cîmp

Deplasarea ferestrei în sus şi în jos se face prin ↑R şi ↑C.

RESET [disc]

Recitire tabel de alocare a unui disc după schimbarea volumului din unitatea curentă sau din unitatea <disc>.

HELP

Afişare informaţii generale sau specifice despre DBASE; cuvîntul HELP poate fi urmat de numele unei comenzi sau de un alt cuvînt cheie. Pentru comenzi se afişează sintaxa şi o scurtă descriere.

QUIT [TO comenzi]

Ieşire din DBASE în CP/M după închiderea fişierelor deschise.

12.2.2. Comenzi de prelucrare, actualizare şi de raportare

Afişarea articolelor unui fişier în ordinea valorilor dintr-un cîmp sau din mai multe cîmpuri se poate face fie prin crearea unui alt fişier

ordonat (folosind comanda SORT), fie prin indexarea fişierului, folosind comanda INDEX.

Utilizarea de fişiere index prezintă mai multe avantaje, printre care căutarea mai rapidă după cheia de indexare, folosind comanda FIND, şi utilizarea unor comenzi de actualizare rapidă.

Un fişier de date poate avea asociate mai multe fişiere index, dar numai unul singur este actualizat automat (dacă este activat la deschidere cu USE..INDEX..); de aceea este uneori necesară actualizarea explicită a unor fişiere index prin comanda REINDEX.

Comenzile COUNT şi SUM permit două prelucrări uzuale asupra articolelor selectate: numărarea articolelor care satisfac o condiţie dată (COUNT) şi însumarea valorilor dintr-un cîmp numeric sau mai multe (SUM).

Comanda TOTAL însumează valorile din unul sau cîteva cîmpuri numerice din fiecare grup de articole care au o cheie comună (valoarea dintr-un cîmp sau o combinaţie de cîmpuri) şi creează un nou fişier cu articolele totalizatoare (face subtotaluri pe grupe de articole).

Prelucrarea datelor din două fişiere se poate face prin reunirea lor într-un singur fişier cu articole mai lungi (comanda JOIN) sau prin deschiderea ambelor fişiere şi comutarea alternativă între cele două fişiere simultan deschise prin comanda SELECT.

Actualizarea datelor dintr-un fişier se poate face prin mai multe comenzi, în funcţie de tipul actualizării: CHANGE permite modificarea selectivă a unor cîmpuri în mod interactiv, REPLACE înlocuieşte conţinutul unor cîmpuri cu alte valori rezultate din expresii DBASE, UPDATE înlocuieşte sau adună la conţinutul unor cîmpuri valori extrase dintr-un alt fişier.

Modificarea structurii unui fişier se poate face prin crearea unei alte structuri sau prin editarea structurii existente, folosind comanda MODIFY STRUCTURE.

Prin raportare se înţelege întocmirea unor rapoarte, afişate la imprimantă sau la consolă, pe baza datelor extrase din unul sau mai multe fişiere, eventual prelucrate, sub o formă uşor de utilizat (cu titluri, subtitluri pe pagină, nume de coloane, numerotare pagini, totaluri şi subtotaluri pe pagini etc.). Pentru crearea şi folosirea fişierelor formular de raportare la prezentarea datelor a fost concepută comanda REPORT.

Urmează descrierea comenzilor grupate în acest paragraf.

SORT ON cîmp TO fişier [ASCENDING] [DESCENDING]
Ordonează (sortează) articolele fişierului curent după valorile din cîmpul <cîmp> şi creează fişierul sortat cu numele <fişier>. Fişierul curent rămîne neschimbat şi deschis.

INDEX ON exp TO fiş. index
Indexează fişierul de date curent după cheia de indexare desemnată

prin expresia <exp> și atribuie numele <fiș. index> fișierului index creat (de tip implicit NDX).

REINDEX
Actualizează fișierele index active corespunzător conținutului fișierului de date curent. Fișierele index de actualizat se stabilesc cu comanda SET INDEX TO sau cu USE...INDEX.

COUNT [domeniu] [FOR exp] [TO var]
Calculează numărul de articole din domeniul specificat care satisfac condiția <exp> și eventual atribuie variabilei <var> acest număr. Domeniul implicit este tot fișierul.

SUM [domeniul] cîmp [,..] [TO var [,...]] [FOR exp]
Calculează suma valorilor dintr-un cîmp numeric al tuturor articolelor din domeniul specificat care satisfac condiția <exp> și eventual atribuie rezultatul unei variabile. O comandă SUM poate calcula pînă la 5 sume în paralel. Domeniul implicit cuprinde toate articolele din fișier care nu sînt marcate pentru ștergere.

TOTAL ON cheie TO fișier [FIELDS cîmp [,...]] [FOR exp]
Totalizează valorile din cîmpurile specificate pentru articolele care satisfac expresia <exp> și care au o cheie comună; creează un nou fișier de date cu articolele rezultate prin totalizare. Fișierul curent trebuie să fie sortat sau indexat după cheia <cheie>, astfel încît totalizarea să se poată face la o singură parcurgere a fișierului.

SELECT [PRIMARY] [SECONDARY]
Selectează fișierul primar sau fișierul secundar ca fișier curent (sau selectează zona de lucru pentru fișierul primar sau secundar).

JOIN TO fișier FOR exp [FIELDS cîmp [,...]]
Alipește articolele din fișierele primar și secundar care satisfac condiția <exp>, eventual cu selecția cîmpurilor specificate după cuvîntul FIELDS din cele două tipuri de articole; creează un nou fișier de date <fișier> cu articolele rezultate din alipire.

CHANGE [domeniul] FIELD cîmp [,...] [FOR exp]
Afișează și permite modificarea selectivă sau introducerea de noi date în cîmpurile specificate pentru articolele din domeniul dat care satisfac expresia <exp>. Domeniul implicit este articolul curent.

REPLACE [domeniu] [FOR exp] cîmp WITH exp
[, cîmp WITH exp, ...]

Înlocuiește valorile din cîmpurile specificate prin expresiile date în comandă pentru articolele din <domeniu> care satisfac expresia <exp>.

Domeniul implicit este articolul curent. Se actualizează automat un fişier index activ.

UPDATE FROM fişier ON cheie [ADD cîmp [,..]]
[REPLACE cîmp [,...]]

Actualizează fişierul curent folosind articolele care au o aceeaşi ⟨cheie⟩ din fişierul ⟨fişier⟩ prin adunare şi (sau) prin înlocuire în cîmpurile specificate. Cheia este un nume de cîmp. Fişierul curent trebuie să fie sortat sau indexat după cîmpul ⟨cheie⟩, iar fişierul ⟨fişier⟩ trebuie să fie ordonat după acelaşi cîmp cheie. Articolele care nu au corespondent în celălalt fişier rămîn neschimbate.

MODIFY STRUCTURE

Permite modificarea structurii fişierului curent prin afişarea şi editarea structurii existente. Datele din fişierul curent se pierd la modificarea structurii şi de aceea trebuie salvate prin copiere în alt fişier înainte de modificarea structurii.

REPORT [domeniul] [FORM fişier] [TO PRINT] [FOR exp]

Afişează un raport cu datele din articolele unui domeniu care satisface o expresie ⟨exp⟩ folosind fişierul formular ⟨fişier⟩ (de tip implicit FRM). Se caută un fişier de tip FRM cu numele ⟨fişier⟩, dacă se găseşte atunci se foloseşte în raportare, iar dacă nu există atunci se iniţiază un dialog la consolă pentru definirea fişierului formular. Afişarea se face la consolă sau la imprimantă, dacă se foloseşte opţiunea TO PRINT.

12.3. Comenzi DBASE utilizate în programe

12.3.1. Comenzi pentru introducere şi afişare date

Comanda de afişare a rezultatului unei expresii (comanda ?) se foloseşte atît în programe, cît şi direct de la consolă, pentru vizualizarea valorilor unor variabile, funcţii, cîmpuri sau pentru afişarea unor texte conţinute în constante de tip şir.

Comanda de poziţionare, afişare şi introducere de date (comanda @) este folosită mai ales în programe, pentru crearea unor formulare de introducere date pe ecran sau pentru crearea de rapoarte la imprimantă. Tot pentru afişare sînt utile comenzile de ştergere ecran (ERASE) şi de avans la pagină nouă (EJECT).

Dacă se foloseşte ca opţiune de citire GET la comanda de poziţionare @, atunci este necesară o altă comandă (READ) care face efectiv citirea şi preluarea datelor. Comenzile ERASE şi CLEAR GET pot anula efectul unor comenzi de citire READ.

Comenzile ACCEPT și INPUT se folosesc pentru introducerea de date după afișarea eventuală a unui mesaj explicativ, permițînd prelucrarea prin programe DBASE mai generale a unor date variabile.

Comanda WAIT poate fi folosită fie pentru citirea unui singur caracter într-o variabilă, fie pentru punerea în așteptare a unui program pînă la tastarea oricărui caracter de la consolă.

În acest paragraf sînt prezentate și comenzile asociate variabilelor de memorie, considerînd că utilizarea lor principală este în programe DBASE, uneori în legătură cu operații de I/E.

O variabilă de memorie poate primi o valoare fie prin citire, fie prin atribuirea rezultatului unei expresii, cu comanda STORE.

Limitarea numărului variabilelor de memorie poate fi depășită prin salvarea într-un fișier (SAVE) și refacerea unui set de variabile (RESTORE), sau prin anularea și reutilizarea unor variabile (comanda RELEASE de eliberare selectivă a unor variabile).

? exp [exp,..]

Afișează la consolă valoarea a una sau mai multe expresii.

@ x,y [SAY exp [USING 'șablon']]
[GET var [PICTURE 'șablon']]

Poziționare pe ecran sau la imprimantă în linia x și coloana y (x și y pot fi expresii numerice), urmată eventual de afișarea valorii unei expresii și (sau) de citirea unei valori pentru o variabilă. Comanda de poziționare fără opțiunile SAY și GET produce pe ecran ștergerea caracterelor din linia x de după poziția y. Dispozitivul unde se face afișarea se stabilește prin comanda SET FORMAT TO SCREEN/PRINTER, implicit fiind ecranul. Variabila care urmează lui GET trebuie să fie inițializată printr-o comandă STORE anterioară pentru validarea valorii introduse pentru această variabilă și pentru determinarea formatului implicit de introducere (în absența lui PICTURE). Cuvintele USING și PICTURE preced un șablon sub forma unui șir de caractere între ghilimele, care dictează formatul de afișare și respectiv de introducere, inclusiv anumite verificări la introducere. Șablonul poate conține următoarele caractere:

\# sau 9 afișare (introducere) de cifre zecimale.

La introducere sînt permise și caracterele următoare
„," „." „;" „:" „," „+" „," „," „—"

A afișare (introducere) de litere

X afișare (introducere) orice caractere

$ sau * afișare de „$" sau „*" în locul zerourilor inițiale ale unui număr

! transformare din litere mici în litere mari

Orice alt caracter prezent în șablon este afișat ca atare, atît la USING cît și la PICTURE. Citirea valorii variabilei precedente de GET se face la întîlnirea unei comenzi READ, astfel încît este posibil să se afișeze mai multe linii pe ecran printr-o secvență de comenzi „@" (un formular), urmînd ca introducerea de date să aibă loc după afișarea

întregului formular cu o comandă READ pentru toate variabilele GET (pot fi maxim 64 de comenzi @...GET în aşteptare).

READ

Permite introducerea sau modificarea de date în mod ecran, folosind caracterele de control de poziţionare şi de editare de la comanda EDIT, pentru variabile din comenzi @ cu opţiunea GET. Cursorul poate fi mutat de la o variabilă la alta. Valorile introduse modifică automat variabilele de memorie sau cîmpurile afişate. O comandă READ se aplică pentru toate comenzile @...GET anterioare.

ERASE

Şterge ecranul consolei şi anulează comenzile @...GET în aşteptare de date.

EJECT

Avans la începutul paginii următoare, la imprimantă.

CLEAR [GET]

Cu opţiunea GET (sau GETS) se anulează toate cererile de introducere provenite din comenzi @...GET, fără modificarea ecranului. Fără opţiune, CLEAR închide toate fişierele deschise, eliberează toate variabilele de memorie şi selectează zona primară.

ACCEPT ["text"] TO var

Citeşte un şir de caractere de la consolă în variabila <var>, eventual după afişarea unui text. Şirul introdus nu trebuie încadrat de ghilimele.

INPUT ["text"] TO var

Citeşte o valoare de la consolă în variabila <var>, eventual după afişarea unui text. Tipul variabilei rezultă din forma datelor introduse, de aceea şirurile de caractere trebuie încadrate între ghilimele.

WAIT [TO var]

Aşteaptă introducerea unui caracter de la consolă şi, opţional, introduce caracterul citit în variabila <var>.

STORE exp TO var

Memorează valoarea expresiei <exp> în variabila de memorie <var>.

SAVE TO fişier

Salvează toate variabilele de memorie existente în fişierul <fişier> de tip implicit MEM.

RESTORE FROM fişier

Citeşte fişierul cu variabile de memorie <fişier> (de tip MEM) şi creează variabilele corespunzătoare, anulînd variabilele anterioare.

RELEASE [var,...] [ALL]

Eliberează variabilele de memorie din lista de variabile specificate sau toate variabilele, cu opţiunea ALL.

12.3.2. Comenzi de control şi alte comenzi pentru programare

Orice limbaj de programare trebuie să permită scrierea de programe neliniare, cu decizii şi cicluri, prin instrucţiuni de control a succesiunii de execuţie a comenzilor de prelucrare.

Limbajul DBASE este complet structurat, în sensul că nu există comenzi de salt în program (şi nici posibilitatea de etichetare a comenzilor), fiind prevăzute comenzi pentru programarea a trei structuri de control: decizia binară sau selecţia între două alternative (IF), ciclul cu condiţie la început (DO WHILE) şi selecţia între mai multe cazuri (DO CASE).

Fiecare dintre aceste structuri este delimitată printr-o comandă iniţială (IF, DO WHILE, DO CASE) şi o comandă finală (ENDIF, ENDDO), între care se pot afla oricîte comenzi (inclusiv comenzi de control).

Comanda ELSE este necesară pentru introducerea alternativei de condiţie neîndeplinită la IF, iar comanda LOOP permite un salt la sfîrşitul unui ciclu DO WHILE, pentru continuarea ciclului.

Limbajul DBASE permite crearea de programe modulare, dar un modul de program este în DBASE un fişier de comenzi separat; apelarea unui modul pentru execuţie prin comanda DO este întotdeauna precedată de încărcarea în memorie a fişierului de comenzi apelat. În acest fel, fişierele de comenzi DBASE corespund atît subrutinelor (procedurilor) din alte limbaje, cît şi segmentelor de program încărcate succesiv în memorie.

Revenirea dintr-un fişier de comenzi în fişierul care l-a apelat şi lansat se face prin comanda RETURN, iar terminarea întregului program din orice fişier de comenzi se face prin comanda CANCEL.

În acest paragraf mai sînt prezentate comenzile pentru introducerea de comentarii în programe DBASE, precum şi comanda MODIFY COMMAND folosită în crearea şi modificarea fişierelor de comenzi (practic un editor de texte).

IF exp
... (comenzi)
[ELSE]
[...] (comenzi)
ENDIF

Testează valoarea expresiei logice ⟨exp⟩; dacă ⟨exp⟩ este adevărată, atunci se execută toate comenzile următoare pînă la o comandă ELSE sau ENDIF; dacă expresia ⟨exp⟩ este falsă, atunci se execută comenzile dintre ELSE şi ENDIF (dacă există ELSE) sau se trece la comanda imediat următoare lui ENDIF.

DO WHILE exp
... (comenzi)
ENDDO

Execută în mod repetat comenzile dintre DO WHILE și ENDDO, atît timp cît este adevărată condiția ⟨exp⟩. Dacă ⟨exp⟩ este falsă de la început, atunci comenzile nu se execută niciodată.

LOOP

Salt peste comenzile următoare, la comanda ENDDO pentru continuarea ciclului prin testarea expresiei din DO WHILE.

DO CASE
CASE exp1
... (comenzi)
CASE exp2
... (comenzi)
... (alte cazuri)
[OTHERWISE]
... (comenzi)
ENDCASE

Selectează cazul pentru care este adevărată expresia ⟨expi⟩, prin salt la secvența de comenzi ce urmează liniei CASE respective. Dacă nu este adevărată nici una dintre expresii, atunci se execută secvența care urmează cuvîntului OTHERWISE.

DO fișier

Încarcă în memorie și execută fișierul de comenzi cu numele ⟨fișier⟩, cu tipul implicit CMD. Este posibilă și lansarea în execuție a unui fișier de comenzi imediat după încărcarea programului DBASE, dacă se folosește comanda „DBASE fișier".

RETURN

Termină execuția fișierului de comenzi activ și revine în fișierul de comenzi de unde a fost apelat sau în DBASE.

CANCEL

Termină execuția unui fișier de comenzi și revine în DBASE.

NOTE text
* text

Introduce un text comentariu, ignorat și neafișat la executarea fișierului de comenzi.

REMARK text

Introduce un comentariu, neinterpretat, dar afișat la executarea fișierului de comenzi.

MODIFY COMMAND fișier

Activează un editor de texte simplu în mod ecran, incorporat în DBASE și destinat în principal pentru crearea sau modificarea fișierelor de

comenzi sau altor fişiere DBASE (tipul implicit pentru ⟨fişier⟩ este CMD). Ieşirea din editor se face prin CTRL/W, cu păstrarea versiunii anterioare într-un fişier de tip BAK, sau cu CTRL/Q care abandonează editarea şi lasă fişierul de intrare neschimbat. Editorul recunoaşte următoarele caractere de control:

↑ E, ↑ A	mută cursor pe linia precedentă
↑ X, ↑ F	mută cursor pe linia următoare
↑ S	mută cursor pe caracterul precedent
↑ D	mută cursor pe caracterul următor
↑ G	şterge caracterul din dreptul cursorului
↑ Y	şterge linia curentă cu blancuri
↑ T	şterge şi elimină de pe ecran linia curentă
↑ N	mută în jos liniile următoare pentru inserare de linie
↑ R	mută ecranul în sus cu o linie
↑ C	mută ecranul în jos cu o pagină
↑ V	comutare între mod înlocuire şi mod inserare

Limitările editorului de texte DBASE sînt următoarele: liniile de text pot avea cel mult 77 de caractere (inclusiv CR, LF), caracterul TAB este înlocuit cu un spaţiu, cursorul nu poate fi mutat înapoi în fişier cu mai mult de 4 000 caractere.

Exemple de utilizare a comenzilor de control:

1) Determinarea valorii maxime din cîmpul VAL al tuturor articolelor fişierului curent

```
GO TOP
STORE VAL TO MAX
DO WHILE .NOT. EOF
  SKIP
  IF MAX >= VAL
   LOOP
  ELSE
   STORE VAL TO MAX
  ENDIF
ENDDO
```

2) Determinarea valorii maxime din cîmpul VAL al tuturor articolelor din fişierul curent, folosind LOOP (varianta inferioară, nerecomandată)

```
GO TOP
STORE VAL TO MAX
DO WHILE .NOT. EOF
  SKIP
  IF MAX >= VAL
   LOOP
  ELSE
   STORE VAL TO MAX
  ENDIF
ENDDO
```

β) Selectarea execuţiei unui fişier de comenzi în funcţie de valoarea unui caracter citit

```
WAIT TO OPTION
DO CASE
 CASE OPTION='L'
  DO LISTARE
 CASE OPTION='E'
  DO EDITARE
 OTHERWISE
  DO LISTOPT
ENDCASE
```

12.3.3. Comanda SET

Comanda SET permite modificarea unor opţiuni de lucru ale programului DBASE şi este folosită mai ales în fişiere de comenzi.

Există două forme ale comenzii SET, după cum parametrul modificat poate avea două valori (ON/OFF = activ/inactiv) sau poate avea ca valoare un şir oarecare (de exemplu un nume de fişier) sau un cuvînt cheie:

SET param [ON] [OFF]
SET param TO opţiune

Urmează descrierea efectului fiecărei comenzi SET, pentru parametrul activ (ON) la formele cu două valori posibile.

SET **ECHO** [ON] [OFF]
Afişează comenzile dintr-un fişier pe măsură ce sînt interpretate (implicit OFF). Se foloseşte pentru depanarea programelor DBASE.

SET **STEP** [ON] [OFF]
Opreşte execuţia unui fişier de comenzi după fiecare comandă, permiţînd utilizatorului să continue cu comanda următoare, să introducă o comandă de la tastatură sau să iasă din fişierul respectiv. Se foloseşte pentru depanarea programelor DBASE (implicit OFF).

SET **TALK** [ON] [OFF]
Afişează sau nu pe ecran rezultatul unor comenzi (implicit OFF)

SET **PRINT** [ON] [OFF]
Afişează şi la imprimantă ceea ce se afişează pe ecran (implicit OFF).

SET **CONSOLE** [ON] [OFF]
Afişează şi la consolă efectul comenzilor (implicit ON)

SET **ALTERNATE** [ON] [OFF]
Scrie pe disc ceea ce se afişează pe ecran (implicit OFF). Numele fişierului se stabileşte printr-o comandă SET ALTERNATE TO fişier

SET **SCREEN** [ON] [OFF]
Foloseşte sau nu modul ecran pentru comenzile APPEND, INSERT, EDIT, CREATE (implicit ON)

SET **LINKAGE** [ON] [OFF]
Face ca fişierele primar şi secundar să fie legate (ON) sau independente (OFF); pentru fişierele legate, comenzile care au un domeniu ca parametru realizează poziţionarea în ambele fişiere (implicit OFF)

SET **COLON** [ON] [OFF]
Afişează sau nu caracterul ':' între date diferite citite prin comanda @...GET (implicit ON)

SET **BELL** [ON] [OFF]
Produce sau nu un semnal sonor la introducerea de date eronate sau la depăşirea lungimii unui cîmp (implicit ON).

SET **ESCAPE** [ON] [OFF]
Permite terminarea forţată a unui fişier de comenzi prin caracterul ESC (implicit ON).

SET **EXACT** [ON] [OFF]
Stabileşte modul de comparare a două şiruri de caractere: pe toată lungimea şirurilor comparate (ON) sau numai pe lungimea celui de al doilea şir (implicit OFF).

SET **INTENSITY** [ON] [OFF]
Activează sau inhibă folosirea a două intensităţi diferite la afişare sau afişarea cu video normal şi video invers (implicit ON).

SET **DEBUG** [ON] [OFF]
Se trimite la imprimantă ieşirea comenzilor ECHO şi STEP, pentru a nu apărea pe ecran la depanarea unor comenzi care operează în mod ecran (implicit OFF).

SET **CARRY** [ON] [OFF]
Se preiau automat datele din articolul anterior la adăugarea de noi articole în mod ecran cu APPEND (implicit OFF).

SET **CONFIRM** [ON] [OFF]
Cere confirmarea operatorului, prin apăsarea clapei ⟨CR⟩, pentru a trece la cîmpul următor după completarea cîmpului curent în mod ecran (ON) sau trece automat la cîmpul următor, dacă s-a completat cîmpul curent (OFF) (implicit OFF)

SET **EJECT** [ON] [OFF]
Se trece sau nu la început de pagină nouă înainte de afişarea unui raport cu comanda REPORT (implicit ON)

SET **RAW** [ON] [OFF]
Inserează sau nu spaţii între cîmpurile afişate cu comenzile LIST şi DISPLAY fără listă de cîmpuri (implicit OFF)

SET **HEADING** TO şir
Memorează şirul <şir>, de maximum 60 caractere, pentru a fi imprimat ca parte a titlului de raport.

SET **FORMAT** TO [SCREEN] [PRINT]
Stabileşte dispozitivul la care se afişează rezultatul comenzilor @ (consolă sau imprimantă).

SET **FORMAT** TO fişier
Stabileşte fişierul de tip FMT, care conţine comenzi @ şi care va fi citit şi utilizat la prima comandă READ.

SET **DEFAULT** TO disc
Stabileşte unitatea de disc implicită DBASE pentru fişiere de orice tip; <disc> poate fi una dintre literele A, B,..., urmată sau nu de caracterul ':'.

SET **ALTERNATE** TO fişier
Stabileşte fişierul unde se va scrie ceea ce se afişează în mod normal pe ecran şi deschide acest fişier. Scrierea efectivă în fişier începe însă cu o comandă SET ALTERNATE ON şi poate fi întreruptă cu SET ALTERNATE OFF

SET **DATE** TO zz/ll/aa
Stabileşte sau modifică data curentă (fără nici o validare a datei introduse). Data este cerută de DBASE imediat după lansare, cu excepţia lansării prin comanda „DBASE fişier".

SET **INDEX** TO fişier [,...]
Stabileşte fişierele index active pentru fişierul de date curent. Tipul implicit al fişierelor este NDX.

SET **MARGIN** TO n
Stabileşte numărul <n> de spaţii afişate, înaintea fiecărei linii de raport (marginea din stînga a raportului).

12.4. Tehnici de programare şi exemple de programe DBASE

12.4.1. Realizarea interfeţei dintre aplicaţie şi utilizator

Aplicaţiile care urmează a fi exploatate de către utilizatori neprogramatori pot prezenta pentru aceşti utilizatori o interfaţă mai prietenoasă, bazată mai mult pe selectarea unei opţiuni dintr-o listă

de opțiuni posibile și pe răspunsuri la întrebări puse de program. Dialogul utilizator-aplicație se poate desfășura complet în limba română, inclusiv mesajele de semnalare a unor erori, care în mod normal sînt detectate de către DBASE (de exemplu nume greșit de fișier sau de cîmp).

O asemenea interfață este în general și mai puțin expusă la erori de introducere, deoarece se pot face o serie de validări specifice asupra datelor și asupra parametrilor operației cerute și nu sînt posibile erori de sintaxă la comenzi, deoarece nu există comenzi pentru utilizator.

Lansarea unui asemenea program de aplicație se face printr-o comandă „DBASE fișier", pentru a evita dialogul inițial angajat de DBASE pentru introducerea datei curente.

O interfață cu listă de opțiuni afișează pe ecran o listă de funcții realizate de program și fiecărei funcții i se asociază o cifră sau o literă; urmează citirea de la consolă a unui caracter și selectarea unei secvențe de instrucțiuni în funcție de valoarea caracterului introdus.

După executarea funcției cerute se reafișează lista de opțiuni și se așteaptă un nou caracter de selectare a unei opțiuni; această secvență se repetă pînă cînd utilizatorul introduce opțiunea de terminare și ieșire din program.

În cazul programelor care realizează un număr mai mare de funcții, aceste funcții se pot grupa pe două sau mai multe niveluri, la care corespund mai multe meniuri de opțiuni: un meniu principal care conține grupe mari de funcții și cîte un meniu de nivel inferior pentru fiecare grup din meniul principal, în cadrul căruia se afișează funcțiile din grupul respectiv.

Tratarea unei opțiuni se poate face printr-o secvență de comenzi inclusă în programul de afișare a listei de opțiuni sau se poate reduce la o comandă DO de apelare a unui alt fișier de comenzi. *Exemplu* de programare a unui meniu de opțiuni:

```
    STORE 'O' TO OPT
    DO WHILE OPT # 'X'
 * afisare lista de optiuni
    ERASE
    @  6,8 SAY '1 = UTILITARE'
    @  8,8 SAY '2 = INTRODUCERE / MODIFICARE DATE'
    @ 10,8 SAY '3 = CAUTARE/LISTARE DATE'
    @ 14,8 SAY 'X = IESIRE IN DBASE'
    @ 18,10
    ACCEPT "OPTIUNE" TO OPT
 * tratare optiuni
    ERASE
    DO CASE
```

```
CASE OPT='1'
  DO UTILITARE
CASE OPT='2'
  DO MODIFICARE
CASE OPT='3'
  DO LISTARE
CASE OPT='X'
  RETURN
ENDCASE
ENDDO
```

Programele UTILITARE, MODIFICARE, LISTARE pot afişa la rîndul lor alte meniuri de opţiuni.

Dacă secvenţele de tratare a unor opţiuni nu sînt prea lungi (sub 50 de comenzi), atunci este mai bine ca timp de execuţie să nu se folosească alte fişiere de comenzi pentru tratarea acestor opţiuni.

Mai trebuie observat că, la o comandă DO CASE, timpul de selecţie a unui caz este cu atît mai mare cu cît cazul respectiv apare mai spre sfîrşitul listei de cazuri; este bine de aceea ca opţiunile solicitate mai frecvent să fie plasate la începutul listei de cazuri a comenzii DO CASE.

Pentru realizarea unor funcţii ale unui program de aplicaţie sînt de obicei necesari anumiţi *parametri variabili*, care se introduc de către operator ca răspuns la întrebări adresate de program. Parametrii introduşi sînt atribuiţi unor variabile DBASE, iar valoarea lor înlocuieşte numele variabilelor în comenzile DBASE, care realizează funcţia dorită. Într-un asemenea program comenzile DBASE folosesc intens macrosubstituţia.

Exemplu de programare a unei funcţii de sortare fişier.

```
ACCEPT "Nume fisier" TO Fisier
USE &Fisier
ACCEPT "Nume cimp de sortare" TO Cimp
ACCEPT "Ordine: Crescator/Descrescator (C/D)" TO Ordine
IF Ordine='C'
  STORE "ASCENDING" TO Ordine
ELSE
  STORE "DESCENDNG" TO Ordine
ENDIF
ACCEPT "Nume fisier sortat" TO Fnou
SORT ON &Cimp TO &Fnou &Ordine
```

În acest exemplu utilizatorul introduce parametrul 'ordine de sortare' într-o formă convenabilă lui, iar programul îl transformă apoi în forma cerută de comanda DBASE.

Pentru a evita producerea unor erori datorate introducerii incorecte a parametrilor, erori care ar produce mesaje în limba engleză emise de DBASE, se pot introduce *verificări* asupra acestor parametri cu mesaje în limba română.

Exemplu de tratare a opţiunii de sortare cu verificarea parametrilor introduşi.

```
REMARK          Sortare fisier de date
?
ACCEPT "Nume fisier de sortat" TO Fisier
IF .NOT. FILE ('&Fisier')
   ? 'Nu exista fisierul &Fisier (pe discul implicit)'
   ACCEPT '&Text' TO Nul
   RETURN
ENDIF
USE  &Fisier
ACCEPT "Nume cimp de sortare" TO Cimp
IF TYPE(&Cimp)='U'
   ? 'Nu rexista cimpul &Cimp in fisierul &Fisier'
   ACCEPT '&Text' TO Nul
   RETURN
ENDIF
...
SORT ON &Cimp TO &Fnou·&Ordine
```

Variabila 'Text' este folosită în tratarea mai multor opţiuni, după un mesaj de eroare sau după alte mesaje care trebuie să rămînă un timp pe ecran pentru a fi citite de operator; ea se iniţializează la începutul programului astfel:

STORE 'pentru continuare apăsaţi tasta CR' TO Text

Înainte de afişarea listei de opţiuni, un program poate afişa un titlu şi poate solicita anumite *date cu caracter general*, utilizate ulterior în anumite comenzi: data calendaristică, numele discului suport al fişierelor aplicaţiei sau chiar numele fişierelor aplicaţiei.

Exemplu de secvenţă de la începutul unui program de aplicaţie, în care se solicită introducerea numărului discului, care devine disc implicit pentru comenzile următoare.

```
SET TALK OFF
ERASE
?
?          EVIDENTA CONTRACTELOR DE CERCETARE'
?
RELEASE ALL
?
?
? 'Pe ce unitate de disc (0/1) se afla fisierele?'
STORE ' ' TO Disc
DO WHILE Disc # '0' .AND. Disc # '1'
  ACCEPT 'Disc(0/1)' TO Disc
ENDDO
IF Disc='
  STORE 'A:' TO Disc
ELSE
  STORE 'B:' TO Disc
ENDIF
SET DEFAULT TO &Disc
```

Programele DBASE conţin de obicei la început o comandă SET TALK OFF, care interzice afişarea la consolă a efectului unor comenzi din program pe parcursul execuţiei programului (de exemplu la comenzile STORE de iniţializare a unor variabile).

O altă soluţie de stabilire a discului suport constă în solicitarea numelui fişierului şi căutarea acestui nume pe toate unităţile de discuri cuplate la sistem, dar această soluţie necesită un timp mai mare de execuţie.

Exemplu: fragment dintr-un program pentru determinarea automată a discului suport al unui fişier.

```
STORE 'A' TO Disc
ACCEPT "Nume fisier contracte" TO Fisier
IF .NOT. FILE (&Disc.:&Fisier)
  STORE 'B' TO Disc
  IF .NOT. FILE(&Disc.:&Fisier)
   ? 'Nu exista fisierul &Fisier pe unitatile 0 si 1'
   ACCEPT TO Nul
   RETURN
  ENDIF
ENDIF
USE &Disc.:&Fisier
```

Introducerea datei calendaristice poate fi însoţită de o validare mai simplă sau mai complexă asupra numărului de zile din fiecare lună.

Exemplu: fragment de program pentru introducerea datei cu validare simplă.

```
STORE F TO Corect
STORE '        ' TO Data
DO WHILE .NOT. Corect
 @15,5 SAY 'Data (ZZ/LL/AA)' GET Data PICTURE '99/99/99'
 READ
 IF $(Data,7,2)='87'
  IF VAL($(Data,1,2)) >= 1 .AND. VAL($(Data,1,2)) <= 31
   IF VAL($(Data,4,2)) >= 1 .AND. VAL($(Data,4,2)) <= 12
       STORE T TO Corect
     ENDIF
    ENDIF
  ENDIF
ENDDO
SET DATE TO &Data
```

O verificare mai completă se poate face folosind o listă de variabile cu numărul de zile din fiecare lună şi verificînd numărul zilei în funcţie de numărul lunii.

Exemplu: fragment de program pentru introducere şi validare numărui din data calendaristică.

```
STORE 30 TO LUN04,LUN06,LUN09,LUN11
STORE 31 TO LUN01,LUN03,LUN05,LUN07,LUN08,LUN10,LUN12
STORE 28 TO LUN02
DO WHILE  NOT  Corect
...
STORE VAL($(Data,1,2)) TO zi
STORE $(Data,4,2) TO L
STORE VAL(L) TO luna
STORE VAL($(Data,7,2) TO an
IF an/4.0 = an/4
  STORE 29 TO LUN02
ENDIF
IF an > 85
 IF luna >=1 .AND. luna <= 12
  IF zi >=1 .AND. zi <= LUN&L
   STORE T TO Corect
  ENDIF
 ENDIF
ENDIF
IF .NOT. Corect
 @ 20,20 SAY 'Eroare la data!'
ENDIF
ENDDO
```

Deşi o asemenea validare complexă poate să nu fie justificată ca timp, am ilustrat prin acest exemplu cum se poate realiza în DBASE o indexare într-o listă de variabile, fără să existe noţiunile de mulţime de variabile (tablou) şi de variabilă indexată: toate variabilele mulţimii au începutul numelui identic şi diferă prin ultimele caractere; selectarea unei variabile se face printr-un nume în care ultimele caractere rezultă din substituţia unei variabile (de exemplu LUN & L, unde variabile L poate avea valorile 01, 02,...12).

12.4.2. Actualizarea datelor din fişiere

Introducerea de date într-un fişier se poate face direct prin comanda APPEND, în formatul prevăzut de această comandă şi cu verificări minime asupra lungimii şi tipului fiecărui cîmp.

Dacă însă se doreşte un alt format de introducere şi, mai ales, efectuarea mai multor *teste de validare* a datelor introduse, atunci trebuie scris un program de introducere şi de validare a datelor. Un asemenea program foloseşte comanda APPEND BLANK pentru adăugarea unui articol fără date, urmată de citirea datelor în variabile de memorie şi de o comandă REPLACE, care înlocuieşte conţinutul nul, al ultimului articol adăugat cu valorile variabilelor.

Exemplu: Program de adăugare a mai multor articole la un fişier după validarea lor.

```
USE PERSONAL
STORE T TO repet
DO WHILE repet
 ERASE
 APPEND BLANK
 STORE '               ' TO Vnume
 STORE '         ' TO Vprenume
 @1,10
 ? "Articol nr.",#
 @ 3,1 SAY "Nume" GET Vnume PICTURE 'AAAAAAAAAAAAAAA'
 READ
 @ 5,1 SAY "Prenume" GET Vprenume PICTURE 'AAAAAAAAA'
 READ
 STORE 'X' TO Vsex
 DO WHILE Vsex # 'M' .AND. Vsex # 'F'
  @ 7,1 SAY "Sex (M/F)" GET Vsex
  READ
 ENDDO
 STORE 0 TO Vani
 STORE F TO corect
 DO WHILE .NOT. corect
  @ 9,1 SAY "Virsta in ani" GET Vani PICTURE 99
  READ
  IF Vsex='M' .AND. Vani <= 70 .OR.
     Vsex='F' .AND. Vani <= 60
   STORE T TO corect
  ENDIF
 ENDDO
 REPLACE NUME WITH Vnume, PRENUME WITH Vprenume
 REPLACE SEX WITH Vsex, VIRSTA WITH Vani
 @ 23,10 SAY "Mai adaugati? (Y/N)" GET repet
 READ
ENDDO
```

În acest exemplu avem două tipuri de validări: validare individuală a unui cîmp şi validare corelată între două cîmpuri (SEX şi VÎRSTĂ).

În aplicaţiile de introducere a datelor se doreşte uneori o *formatare mai complexă a ecranului*, care să semene cît mai mult cu un formular imprimat folosit în colectarea de date primare. În acest scop se folosesc comenzi de poziţionare-afişare-citire (@) care pot conţine o descriere mai completă a cîmpurilor decît numele de cîmpuri (limitat la 10 caractere).

Dacă un acelaşi formular este necesar în cîteva programe diferite (de introducere, de vizualizare-corecţie, de căutare, etc.) atunci se recomandă crearea unui fişier format (de tip FMT), care să conţină toate comenzile „@" dintr-un ecran.

În cazul fişierelor indexate după o cheie, trebuie activat şi fişierul index la deschiderea fişierului de date în vederea adăugării de noi date. Dacă s-au făcut adăugări de articole la un fişier fără ca indexul să fie activat sau dacă sînt mai multe fişiere index asociate unui fişier de date, atunci se va folosi comanda REINDEX pentru aducerea fişierelor index în concordanţă cu noul conţinut al fişierului de date.

Indexarea fişierelor este o tehnică de programare mult folosită în aplicaţiile DBASE, datorită avantajelor pe care le oferă:

— căutare rapidă după conţinutul unui cîmp sau după o cheie de indexare care combină valori din mai multe cîmpuri;

— listarea în ordinea valorilor din cîmpul indexat sau în ordinea valorilor cheii de indexare;

— permite menţinerea în ordinea indexului în cursul actualizării fişierului prin adăugări, inserări, ştergeri de articole, fără a fi necesară o operaţie separată de reordonare.

— permite utilizarea unor comenzi performante de actualizare a conţinutului articolelor, care necesită fişiere indexate sau ordonate.

Indexarea constituie o alternativă, de cele mai multe ori preferabilă, la ordonarea unui fişier prin comanda SORT.

Sortarea se foloseşte pentru fişiere temporare, cum ar fi un fişier cu date primare, utilizat în actualizarea unui fişier principal, permanent, sau atunci cînd se cere o listare a unui fişier principal într-o altă ordine decît cea asigurată de index.

Actualizarea conţinutului unui fişier de date se poate face în două moduri:

— prin operarea modificărilor direct în fişier de la consolă, folosind comenzile EDIT sau CHANGE;

— prin colectarea modificărilor într-un fişier, urmată de actualizarea fişierului principal cu datele din fişierul de actualizări, folosind comenzile REPLACE, UPDATE sau TOTAL.

Comanda REPLACE lucrează la nivel de articol, în timp ce comenzile UPDATE şi TOTAL operează la nivel de fişier, mai rapid.

Programul DBASE permite prelucrarea fişierelor articol cu articol, prin cicluri programate cu DO WHILE şi prin trecerea de la un articol la altul cu SKIP, aşa cum se face în alte limbaje de programare (COBOL, FORTRAN ş.a.). În acelaşi timp, în DBASE sînt prevăzute o serie de comenzi specifice sistemelor de baze de date relaţionale, care execută o anumită operaţie asupra tuturor articolelor din fişier sau numai asupra anumitor articole selectate printr-o expresie, realizînd intern ciclul de parcurgere, testare şi prelucrare a fiecărui articol: APPEND FROM, COPY, LIST, JOIN, REPORT, SORT, TOTAL, UPDATE ş.a.

Utilizarea comenzilor DBASE de prelucrare la nivel de fişier conduce la performanţe mult mai bune decît prelucrarea repetată la nivel de articol şi ca urmare se va recurge la asemenea cicluri explicite de

tratare articol cu articol numai atunci cînd nu pot fi folosite comenzile DBASE prevăzute pentru operațiile respective (ceea ce se întîmplă destul de rar).

În general, atunci cînd se realizează aplicații în DBASE, nu trebuie preluate fără discernămînt soluții folosite cu succes în alte limbaje. De exemplu, utilizarea adreselor de legătură („pointeri'') pentru asocierea de date din fișiere diferite nu este recomandabilă în DBASE, asocierea fiind realizată prin conținutul fișierelor care au un cîmp comun. De asemenea, în DBASE nu se poate lucra cu mai mult de două fișiere de date simultan deschise, iar deschiderea și închiderea repetată a unui fișier la fiecare articol prelucrat durează inadmisibil de mult; de aceea este preferabil chiar să se parcurgă complet un fișier de cîteva ori, decît să se parcurgă o singură dată, cu deschidere și închidere la fiecare articol (pentru a nu avea mai mult de două fișiere deschise).

În cazul cînd actualizarea unui fișier constă doar în modificarea conținutului unor cîmpuri din articolele existente, fie prin înlocuire, fie prin adăugarea unor noi valori la cele existente, se poate folosi cu bune performanțe comanda UPDATE.

Exemplu: Un fișier principal cu numele MAIN are mai multe cîmpuri dintre care COD, CANT, VAL, iar un fișier de modificări cu numele UPDT are numai cîmpurile COD, CANT, VAL. Operarea modificărilor conținute în UPDT prin cumularea cu valorile conținute în articolele corespunzătoare (cu același cod) din MAIN se va face astfel:

```
USE UPDT
SORT ON COD TO TEMP
USE MAIN INDEX XCOD
UPDATE FROM TEMP ON COD ADD CANT,VAL
DELETE FILE TEMP
```

Condiția necesară utilizării comenzii UPDATE este ca atît fișierul principal, cît și fișierul de modificări să fie ordonate sau indexate după același cîmp. Cheia care urmează cuvîntului ON la comenzile UPDATE și TOTAL poate fi numai numele unui cîmp și nu o combinație de cîmpuri, ceea ce poate ridica probleme uneori (de exemplu dacă cantitatea și valoarea sînt asociate unei perechi client-produs și nu doar unui client sau unui produs).

În cazul în care actualizarea unui fișier constă atît din modificarea conținutului unor cîmpuri, cît și din adăugarea unor noi articole, atunci se poate folosi comanda TOTAL după concatenarea celor două fișiere prin comanda APPEND.

Exemplu: un fișier principal MAIN conține și cîmpurile COD, CANT, VAL fiind indexat după cîmpul COD; fișierul de modificări UPDT conține doar cîmpurile COD, CANT, VAL și este ordonat după

cîmpul COD. Actualizarea fişierului MAIN cu datele din UPDT se face prin secvenţa de comenzi următoare:

```
USE MAIN INDEX XCOD
APPEND FROM UPDT FIELDS COD,CANT,VAL
TOTAL ON COD TO TEMP FIELDS CANT,VAL
USE
DELETE FILE MAIN
RENAME TEMP TO MAIN
DELETE FILE XCOD.NDX
USE MAIN
INDEX ON COD TO XCOD
```

Programul este ceva mai mare, deoarece nu se poate totaliza în fişierul de intrare MAIN.

Actualizarea printr-un ciclu explicit, folosind comanda REPLACE la fiecare articol modificat, necesită lucrul cu două fişiere simultan deschise şi selectarea alternativă a fiecăruia dintre fişiere ca fişier curent.

Exemplu: Fişierele MAIN şi UPDT din exemplul anterior dar UPDT nu este ordonat; aceeaşi actualizare realizată acolo cu TOTAL.

```
USE MAIN INDEX XMAIN
SELECT SECONDARY
USE UPDT
DO WHILE .NOT. EOF
 STORE COD TO CHEIE
 SELECT PRIMARY
 FIND &CHEIE
 IF # = 0
  * articol nou
  APPEND BLANK
  REPLACE P.COD WITH S.COD
  REPLACE P.CANT WITH S.CANT
  REPLACE P.VAL WITH S.VAL
 ELSE
  * articol existent
  REPLACE P.CANT WITH P.CANT+S.CANT
  REPLACE P.VAL WITH P.VAL+S.VAL
 ENDIF
 SELECT SECONDARY
 SKIP
ENDDO
```

Modificarea structurii unui fişier se face printr-o secvenţă de forma următoare:

```
USE FVECHI
COPY STRUCTURE TO FNOU
USE FNOU
MODIFY STRUCTURE
APPEND FROM FVECHI
DELETE FILE FVECHI
USE
RENAME FNOU TO FVECHI
```

12.4.3. Prelucrarea şi prezentarea datelor din fişiere

Corespunzător tipului datelor, prelucrările pot fi calcule cu date numerice sau prelucrări pe şiruri de caractere.

Comanda STORE corespunde instrucţiunii de atribuire din alte limbaje, deoarece memorează în una sau mai multe variabile de memorie valoarea unei expresii.

Spre deosebire de alte limbaje de programare, în DBASE tipul variabilelor nu se declară explicit şi nici nu rezultă din numele lor, fiind determinat de tipul constantei sau de tipul rezultatului expresiei care se memorează prima dată într-o variabilă. Utilizarea ulterioară a unei variabile, inclusiv citirea altei valori, trebuie să fie conformă cu tipul rezultat din iniţializarea sa. Este însă posibil ca printr-o altă comandă STORE să se modifice tipul variabilei sau, mai exact, să se redefinească un nume de variabilă. Exemplu:

```
STORE 7 TO X        * X de tip N
STORE '#' TO X      * X de tip C
```

Precizia rezultatului unei expresii este determinată de operandul cu precizie maximă. Exemple:

```
STORE 7/3 TO X           * X primeşte valoarea 2
STORE 7/3. TO X          * X primeşte valoarea 2.3
STORE 7.000/3 TO X       * X primeşte valoarea 2.333
```

Reamintim că numărul total de cifre zecimale la partea întreagă şi la partea fracţionară care pot fi păstrate pentru o dată numerică este de 10.

Două operaţii de calcul uzuale asupra datelor dintr-un fişier — numărarea sau însumarea valorilor dintr-un cîmp pentru toate sau o parte din articolele unui fişier — sînt realizate mai rapid prin comenzile COUNT şi SUM decît prin cicluri de prelucrare articol cu articol. Exemplu:

Program care realizează (într-un timp mai lung) efectul comenzii COUNT FOR CANT <100 TO NPD

```
STORE 0 TO NPD
DO WHILE .NOT. EOF
  IF CANT < 100
   STORE NPD+1 TO NPD
  ENDIF
  SKIP
ENDDO
```

Prelucrările pe şiruri de caractere sînt realizate în principal prin funcţii şi prin operatorii de concatenare. O particularitate o prezintă compararea şirurilor de caractere, care în mod implicit nu se face pe lungimea ambelor şiruri, ci pe lungimea şirului mai scurt. De exemplu,

şirul 'ABC' este egal cu şirul 'ABCDEF'. Prin comanda SET EXACT ON se poate cere o comparare exactă a celor două şiruri.

La prelucrarea datelor extrase din *două fişiere* se va proceda astfel:

— Se deschid o singură dată, la început, cele două fişiere, precizînd care este fişier primar şi care este fişier secundar:

```
SELECT PRIMARY            * poate lipsi la început
USE CLIENTI
SELECT SECONDARY
USE PRODUSE
```

Practic nu are nici o importanţă care dintre cele două fişiere este fişier primar şi care este fişier secundar, deoarece nu este favorizat unul faţă de altul în nici o comandă.

— Numele de cîmpuri din cele două fişiere se pot utiliza în expresii, indiferent care dintre ele este fişier curent. Dacă există un acelaşi nume de cîmp în cele două fişiere, atunci el trebuie precedat de „P"., pentru referire la cîmpul din fişierul primar, sau de „S"., pentru referire la cîmpul din fişierul secundar. Exemplu:

```
IF P.COD = S.COD
REPLACE TOTAL WITH TOTAL + VAL
ENDIF
```

— Înaintea comenzilor sau funcţiilor care se referă implicit la un singur fişier (fişierul curent) trebuie activat fişierul respectiv prin comanda SELECT. Exemplu:

```
USE F1
SELE SECO
USE F2
SELE PRIM
DO WHILE .NOT. E
 IF P.COD=S.COD
   ? 'P',#
   SELE SECO
   ? 'S  ',#
   SKIP
   IF EOF
    LOOP
   ENDIF
   SELE PRIM
  ENDIF
  SKIP
ENDDO
RETURN
```

Prezentarea datelor din unul sau două fişiere, eventual împreună cu alte derivate din acestea prin prelucrări, se poate face în mai multe feluri, în funcţie de conţinutul şi forma dorită pentru raportul de întocmit.

Comanda LIST afişează într-un format simplu, fără titluri, un număr de cîmpuri selectate dintr-un fişier;

Comanda REPORT, eventual cu un fişier formular pregătit dinainte, permite în plus faţă de comanda LIST următoarele:

— atribuirea unui titlu întregului raport, care se afişează automat pe fiecare pagină;

— trecerea la pagină nouă, după un număr de linii şi numerotarea automată a paginilor;

— modificarea limitei din stînga a fiecărei linii şi a numărului de linii pe pagină;

— includerea în raport a valorilor memorate în fişiere sau a rezultatului unor expresii, în particular constante sau variabile (de exemplu, printr-un caracter constant se poate face separarea coloanelor de raport prin linii verticale);

— continuarea de pe o linie pe alta a unui şir de caractere;

— alinierea unui titlu la stînga sau la dreapta într-o coloană;

— afişarea de totaluri şi subtotaluri de cîmpuri numerice la fiecare pagină şi pe întregul raport.

Dacă exigenţele asupra raportului ce trebuie prezentat depăşesc posibilităţile comenzii REPORT, atunci se va scrie un program de imprimare linie cu linie a situaţiei dorite prin parcurgerea articolelor unui fişier sau a două fişiere. Deoarece un asemenea program conţine în principal comenzi de poziţionare şi afişare ('@'), se poate folosi un fişier auxiliar de format. Activarea dintr-un fişier de comenzi a unui fişier de format se face prin comanda READ, chiar dacă nu se citeşte nimic de la consolă.

Comenzile de poziţionare permit şi întocmirea unor grafice simple, cum ar fi diagrame cu bare (histograme) prin care se prezintă sintactic şi comparativ mai multe valori numerice. Timpul necesar desenării acestor grafice este însă destul de mare, comparativ cu programe similare din alte limbaje.

Exemplu: Program DBASE pentru desenarea unei histograme cu bare verticale pe baza valorilor dintr-un cîmp numeric, cu scalare automată în funcţie de valoarea maximă.

```
* citire date pentru histograma
     ACCEPT "Nume cimp" TO Cimp
     IF TYPE (&Cimp) # 'N'
      ? 'Cimpul &Cimp nenumeric sau inexistent'
      WAIT
      RETURN
     ENDIF
     INPUT 'Valoarea maxima din cimpul &Cimp' TO Maxim ?
     ACCEPT 'Numarul primului articol' TO Part
     ACCEPT 'Numarul ultimului articol' TO Uart
```

```
* desenare histograma
     ERASE
     ? 'Histograma valorilor din cimpul &Cimp'
     GOTO &Part
     STORE 2 TO J
     DO WHILE # <= &Uart
      @ J,0 SAY STR(INT(&Cimp),9)
      STORE &Cimp/Maxim*70 TO N
      STORE 1 TO I
      DO WHILE I <= N
       @ J,I+9  SAY '#'
       STORE I+1 TO I
      ENDDO
      STORE J+1 TO J
      SKIP
     ENDDO
```

BIBLIOGRAFIE SELECTIVĂ

1. Toacşe Gh., *Introducere în Microprocesoare*, Editura Ştiinţifică şi Enciclopedică, Bucureşti, 1986.
2. Petrescu A. ş.a., *Microcalculatoarele Felix M18, M18B, M118*, Editura Tehnică, Bucureşti, 1984.
3. Lupu C., Stăncescu I., *Microprocesoare. Circuite şi Proiectare*, Editura Militară, Bucureşti, 1986.
4. Petrescu A. ş.a., *Totul despre aMIC*, Editura Tehnică, Bucureşti, 1985.
5. Lupu C., Ţepelea V., Purice E., *Microprocesoare — aplicatii*, Editura Militară, Bucureşti, 1982.
6. Căpăţînă O., Haşegan M., Puşcă M., *Proiectarea cu microprocesoare*, Editura Dacia, Cluj-Napoca, 1983.
7. * * * *Sistemul de operare CP/M implementat pe TPD şi JUNIOR*, Întreprinderea de Echipamente Periferice Bucureşti, 1986.
8. * * * *CP/M — Manual de utilizare*, Întreprinderea de Calculatoare Electronice Bucureşti, 1985.
9. * * * *MADS — Sistem de dezvoltare cu aplicaţii multiple*, Întreprinderea Microelectronica Bucureşti, 1985.

Redactor : **VIRGIL SPULBER**
Tehnoredactor : **ANGELA ILOVAN**
Coli de tipar : **24**
Bun de tipar : **3.04.1989**

I.P. Informaţia
Comanda nr. 918
Republica Socialistă România